

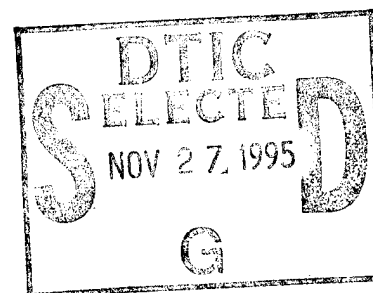


NRL/MR/5521--95-7792

Voice Management and Multiplexing Protocols Developed for the Data and Voice Integration Advanced Technology Demonstration

JAMES P. HAUSER

*Communications Systems Branch
Information Technology Division*



November 13, 1995

19951120 134

DTIC QUALITY INSPECTED 8

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188																	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.																				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 13, 1995	3. REPORT TYPE AND DATES COVERED																		
4. TITLE AND SUBTITLE Voice Management and Multiplexing Protocols Developed for the Data and Voice Integration Advanced Technology Demonstration		5. FUNDING NUMBERS PE-0603792N 3995WXE8011																		
6. AUTHOR(S) James P. Hauser																				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5320		8. PERFORMING ORGANIZATION REPORT NUMBER NRL/MR/5521-95-7792																		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Space and Naval Warfare System Command Washington, DC 20375-5100		10. SPONSORING/MONITORING AGENCY REPORT NUMBER																		
		<table border="1"> <tr> <td colspan="2">Accession For</td> </tr> <tr> <td>NTIS CRA&I</td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>DTIC TAB</td> <td><input type="checkbox"/></td> </tr> <tr> <td>Unannounced</td> <td><input type="checkbox"/></td> </tr> <tr> <td colspan="2">Justification</td> </tr> <tr> <td colspan="2">By</td> </tr> <tr> <td colspan="2">Distribution /</td> </tr> <tr> <td colspan="2">Availability Codes</td> </tr> </table>			Accession For		NTIS CRA&I	<input checked="" type="checkbox"/>	DTIC TAB	<input type="checkbox"/>	Unannounced	<input type="checkbox"/>	Justification		By		Distribution /		Availability Codes	
Accession For																				
NTIS CRA&I	<input checked="" type="checkbox"/>																			
DTIC TAB	<input type="checkbox"/>																			
Unannounced	<input type="checkbox"/>																			
Justification																				
By																				
Distribution /																				
Availability Codes																				
11. SUPPLEMENTARY NOTES																				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		<table border="1"> <tr> <td>Dist</td> <td>Avail and/or Special</td> </tr> <tr> <td>A-1</td> <td></td> </tr> </table>		Dist	Avail and/or Special	A-1		12b. DISTRIBUTION CODE												
Dist	Avail and/or Special																			
A-1																				
13. ABSTRACT (Maximum 200 words) The Data and Voice Integration ATD is a four year program managed by the Naval Research Laboratory (NRL) with a goal of demonstrating integrated communication services using low bandwidth, tactical communication channels. At present, Phases I and II of a three phase demonstration have been completed. Phase I integrated real-time voice and non-real-time data services over a single link, low capacity, tactical circuit. Phase II demonstrated these services over a tactical broadcast network. Phase III will demonstrate data/voice integration with internetworking and multicast routing capabilities. This report focuses on two of the major areas of development critical to the success of this ATD. The first area is the development of voice management protocols for half and full duplex operation. The second is the development of a Data/Voice Integrator (DVI). The DVI design includes a Subnetwork Provider Interface (SNPI), an interface to a Red Link Controller (RLC), queue management, a multiplexing scheme, and a software design. The multiplexing protocol creates a unique form of "link-sharing," e.i., the ability to dynamically divert bandwidth that is reserved but not currently being used to support other services.																				
14. SUBJECT TERMS Data/voice integration Link-sharing Integrated services Voice management protocols			15. NUMBER OF PAGES 54																	
			16. PRICE CODE																	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL																	

Contents

1.0	Scope	1
1.1	Project Status	1
1.2	Document Overview	3
2.0	Applicable Documents	4
3.0	High Level Design	5
3.1	Phase I	5
3.2	Phase II	5
3.3	Phase III	6
3.4	System Components	7
3.4.1	Tactical Voice Terminal	7
3.4.2	IVOX Terminal	8
3.4.3	Tactical Data Terminal	8
3.4.4	JMCIS Terminal	8
3.4.5	System Manager and Boot (SM)	9
3.4.6	Data/Voice Integrator (DVI)	9
3.4.7	MCA Network (MCA)	9
3.4.8	Red Link Controller (RLC)	9
3.4.9	VME Embedded Encryption Module (VEEM)	9
3.4.10	KG-84	10
3.4.11	Black Link Controller (BLC)	10
3.4.12	GPS	10
3.4.13	UHF SATCOM Subsystem	10
3.4.14	HF Subsystem	10
4.0	Tactical Voice Terminal Design	11
4.1	TVT System Behavior	11
4.2	TVT/TVT Messages	17
5.0	Data/Voice Integrator Design	19
5.1	Overview	19
5.2	Services Required from RLC	19
5.3	Half Duplex Circuit Control	19
5.4	DVI Messages	20
5.5	DVI Service Interface	25
5.5.1	DVI Server Access Via TCP	25
5.5.2	DVI as a "TCP Bridge"	26
5.5.3	Client Registration	27
5.5.4	Opening and Closing a DVI Service	27
5.5.5	Datagram Service	27
5.5.6	Virtual Circuit Service	29
5.5.7	Half Duplex Voice Service	29
5.5.8	Flush	30
5.5.9	DVI Status	30
5.6	DVI Operation	33
5.6.1	Initialization Phase	33
5.6.2	Misinterpreted EOTs	34

5.6.3	Operational Phase (Cell Packing, Multiplexing, & Data/Voice Integration)	34
5.7	DVI Software Design	37
5.8	DVI Virtual Circuit Management	40
6.0	Appendix	42
6.1	Virtual Circuit Stream Service	42
6.2	Subnetwork Provider Interface (SNPI) Overview	43
6.3	Phase II (SNPI-based) Virtual Circuit Management	45

List of Figures

1. ATD Phase I system block diagram for a field demonstration node	5
2. ATD Phase II system block diagram for a field demonstration node	6
3. ATD Phase III system block diagram for a field demonstration node	7
4. Expanded state machine diagram of TVT half duplex operation	12
5. Expanded state machine diagram for TVT full duplex operation	13
6. Half duplex call management protocol.	14
7. TVT message format	17
8. DVI state machine diagram.	20
9. DVI peer-to-peer message format.	21
10. DVI datagram and virtual circuit cell and message formats.	23
11. Phase I configuration using a pair of DVIs acting as a "TCP bridge."	26
12. Phase III configuration showing a pair of CS gateways "tunneling" IP via DVI and MCA transport services.	26
13. Commands available from VxWorks rlogin shell for displaying/resetting statistical variables and turning tracing on/off.	31
14. Flow chart of DVI half duplex operation.	33
15. Conceptual view of multiplexed cell streams to form a DVI output stream.	35
16. Packing of Virtual Circuit Message and/or Datagram Cells	35
17. Cell creation and queuing.	36
18. Multiplexing cells to create a transmitted cell stream.	36
19. DVI software design.	39
20. Setup, operation, and termination of a simplex virtual circuit.	41
21. Setup, operation, and termination of a full duplex virtual circuit.	41
22. Cell format for a virtual circuit stream.	42
23. Setup, operation, and termination of a SNPI-based simplex virtual circuit.	46
24. Setup, operation, and termination of a SNPI-based full duplex virtual circuit.	46
25. Setup, operation, and termination of a SNPI-based half duplex virtual circuit.	47

List of Tables

I.	TVT State Machine Events	13
II.	TVT State Machine States	16
III.	TVT End/End Protocol Messages	18
IV.	DVI/MCA Service Requirements	21
V.	Types of Virtual Circuit Control Cells	24
VI.	Messages for Client Registration	27
VII.	Messages for Opening and Closing DVI Services	28
VIII.	Messages for Managing DVI Datagram Services	29
IX.	Messages for Managing DVI Virtual Circuit Services	30
X.	Messages for Half Duplex Voice Control of a DVI	30
XI.	Flush Message	31
XII.	Messages for Requesting and Sending DVI Status	32
XIII.	Virtual Circuit Stream Service Requirements	42
XIV.	SNPI Messages	43

Voice Management and Multiplexing Protocols Developed for the Data and Voice Integration Advanced Technology Demonstration

1.0 Scope

1.1 Project Status

The Data and Voice Integration ATD is a four year program managed by the Naval Research Laboratory (NRL) with a goal of demonstrating integrated communication services using low bandwidth, tactical communication channels. There are three phases of demonstration planned under the program with each phase involving increasing complexity. The demonstrations are as follows: Phase I consists of a single link demonstration that integrates low data rate (1200 bps), real-time voice and non-real-time data services over a low capacity tactical circuit. Phase II involves a tactical network demonstration of data and voice (600, 800, and 1200 bps) integration enabled by new networking protocols operating within a multichannel architecture. Phase III demonstrates data/voice integration with internetworking and multicast routing capabilities.

The Phase I demonstration goal was operation of integrated tactical data and voice services over a single 2400 bps tactical HF groundwave circuit. This goal was accomplished by establishing a circuit spanning the Chesapeake Bay between NRL facilities at Chesapeake Beach, MD, and Tilghman Is., MD, in December, 1993, and by successfully supporting simultaneous data and voice transmissions. The Phase I Data/Voice Integrator (DVI) directly supported several data applications, including a whiteboard, reliable JPEG image file transfer, remote writing (e.g., vtalk), etc. Also, a 1200 bps voice application was demonstrated.

At the present time we have successfully completed both a laboratory demonstration and a field test of the of the HF network developed for Phase II. In mid-June, 1995, NRL personnel conducted a series of four laboratory demonstrations for the Navy using a fully implemented three node HF radio network using closely-coupled, low power antennas to radiate RF energy. In mid-July, 1995, a set of field tests of the same system were successfully concluded. For the field demonstrations, nodes were located at the Naval Academy, at Tilghman Is., MD, and at Chesapeake Beach, MD. As a testimony to its utility, much of the voice coordination during the field tests was done via the HF radio network. Moreover, when it was discovered that we had mistakenly taken an outdated version of one of the test applications to the field, the test coordinator, while stationed at Tilghman, Is., was able to download the current version of the software from his computer at NRL via HF from Tilghman Is. to Chesapeake Beach and, from there, via a phone hookup to NRL. He then disseminated the software to the other sites via HF. Also, test results that were archived at each of the sites on a daily basis were downloaded to the Tilghman Is. site via HF so that preliminary analysis could be performed while still in the field. These test results are now undergoing a thorough analysis and will be the subject of a forthcoming report.

The purpose of this report is to document voice management and multiplexing protocols initially developed and implemented for Phase I of the Data/Voice ATD. Subsequently, with minor modifications, the cell multiplexing technique described in this report has been integrated with Multichannel Architecture (MCA) network software [1, 2] that was used for the Phase II demonstrations.

MCA implements many unique networking concepts to create a robust, adaptive, tactical broadcast network that uses distributed control mechanisms. MCA protocols periodically probe the network to learn network topology and adapt when links or nodes are either lost or recovered. MCA provides relaying capability in several ways. For broadcast services, MCA develops and maintains a broadcast backbone that relays all broadcast traffic. The backbone is reconfigured whenever topology changes occur. Point-to-point services are routed off the backbone, if possible, in order to keep the backbone capacity available for the broadcast services. MCA can support relaying via the backbone, even in a sparsely connected network, without sacrificing broadcast capacity. In a TDMA network, broadcast capacity drops dramatically if it is required to use bandwidth to support relaying in order to maintain network connectivity. MCA maintains capacity by assigning a different channel to each transmitter (one transmitter per MCA node) and allowing each transmitter to transmit in every TDMA slot. Each MCA node has multiple receivers that can be retuned for every slot, if necessary, in order to listen to several neighboring nodes simultaneously.

MCA uses much of the design and the software developed for the DVI as described in this report. It does so in order to add two additional capabilities to MCA. One is to provide an interface that the Internet Layer can use to acquire services from the Subnetwork Layer. This interface has been termed the Subnetwork Provider Interface (SNPI) and is an enhancement of the DVI interface developed for the Phase I demonstration. The other capability added to MCA is that provided by the DVI cell multiplexing protocol. This protocol supports the multiplexing of virtual circuits and datagram services. Virtual circuits exhibit low latency and guaranteed capacity. Capacity that is not reserved, as well as capacity that is reserved but not actually being used, is dynamically allocated to support datagram services. In addition, datagram services can be prioritized.

The Phase II demonstration system also incorporates a Communication Server (CS) component that acts as an IP router and can use MCA's SNPI to acquire subnetwork services on behalf of its users. Users no longer attach directly to the Subnetwork Controller (SC) as was done in the Phase I demonstration system. Instead, users may run standard IP applications that use the CS just as they would use any other IP router or, if a user application has special communication requirements, e.g., virtual circuit quality of service (QOS), an Application Program Interface (API) has been implemented to allow it to negotiate with the CS to accommodate its QOS requirements.

The Phase III demonstration system will add an additional subnetwork to the Phase II system so that the CS can be used to demonstrate the capability to route traffic via multiple wireless subnetworks. A UHF SATCOM link integrated with the new SNPI version of the DVI will provide the second SC.

Other major areas of development for Phase III, besides adding a second subnetwork, will be inclusion of multicast routing (M-OSPF) and resource reservation (RSVP) protocols. The decision to integrate RSVP is a significant one since it will supplant the API recently developed for Phase II. At the time we made the decision to develop the API, RSVP had not reached a sufficient level of maturity to make it a viable option for the Phase II system. However, RSVP now appears to be sufficiently developed to make it usable for Phase III system development. The reason to

migrate to RSVP is to maintain the highest possible level of interoperability with IP-based systems.

1.2 Document Overview

As previously stated, the purpose of this report is to document the voice management and multiplexing protocols developed for Phase I of the Data/Voice ATD (D/V ATD). The multiplexing protocol employs a cell-based technique that supports virtual circuit and datagram services over error-prone, low-bandwidth, tactical communication links. This protocol, along with a service interface was implemented on top of the VxWorks multi-tasking operating system using C++ and has been named the "Data/Voice Integrator" (DVI). As eluded to earlier, the Phase II MCA network has inherited much of the DVI development. MCA modifies some of the message formats described in this report in order to accommodate relaying and will be described in a report on MCA development for Phase II.

This document begins with the presentation of a high level design for all three Phases of the Data/Voice Integration ATD in order to describe the context in which the detailed design of the DVI fits. Also, voice management protocol designs for a Tactical Voice Terminal (TVT) are developed as part of the DVI context. Management of half duplex voice places special requirements upon the Subnet Controller Interface (SNPI) supported by the DVI and, therefore, is a prerequisite to defining that interface.

The primary focus of this report is the design of the DVI component. The DVI served as the development vehicle for the SNPI and the implementation of virtual circuit and datagram services that can be accessed via the SNPI. The predecessor to the SNPI, as it was implemented for Phase I of the D/V ATD, is documented in the body of this report and is the only documentation that exists. The SNPI, which is essentially an embellishment of the original DVI interface, is described in an appendix. Full SNPI documentation is given in [3]. At the heart of the DVI design are the message and cell formats used to create DVI virtual circuit and datagram services and the cell multiplexing scheme used to control bandwidth usage.

2.0 Applicable Documents

1. D. J. Baker, J. P. Hauser, D. N. McGregor, and W. A. Thoet, "Design and Performance of an HF Multichannel Intratask Force Communication Network," NRL Report 9322, November 12, 1991.
2. W. A. Thoet, D. J. Baker, and D. N. McGregor, "A Multichannel Architecture for Naval Task Force Communication," NRL/FR/5520--94-9703, January 30, 1994.
3. "Software Requirements Specification for the Data and Voice Integration Advanced Technology Demonstration," ViaSat Document # 608300-94-001, April, 25, 1994.
4. J. P. Hauser, "System Design and Development of a Low Data Rate Voice (1200 bps) Rate Converter," NRL/MR/5520-92-7136, September 30, 1992.

3.0 High Level Design

3.1 Phase 1

Figure 1 shows a Phase I system design for a single ATD node and indicates a mapping of node components to the OSI architecture. The Application Layer consists of multiple user processes that remotely access the DVI server via TCP connections across an Ethernet LAN. User processes may execute on one or several host terminals. For the Phase I field demonstration in December, 1993, a Sun Workstation was used to host several datagram applications plus the System Manager & Boot, while a PC w/ I860 processor board hosted a Tactical Voice Terminal. The Data/Voice Integrator and Red Link Controller shared a MVME167-34A single board computer running the VxWorks 5.1 real-time operating system. DataComm, Inc., Model 1102 serial tone modems were employed to support an HF groundwave link that carried the cell stream supplied by the DVI.

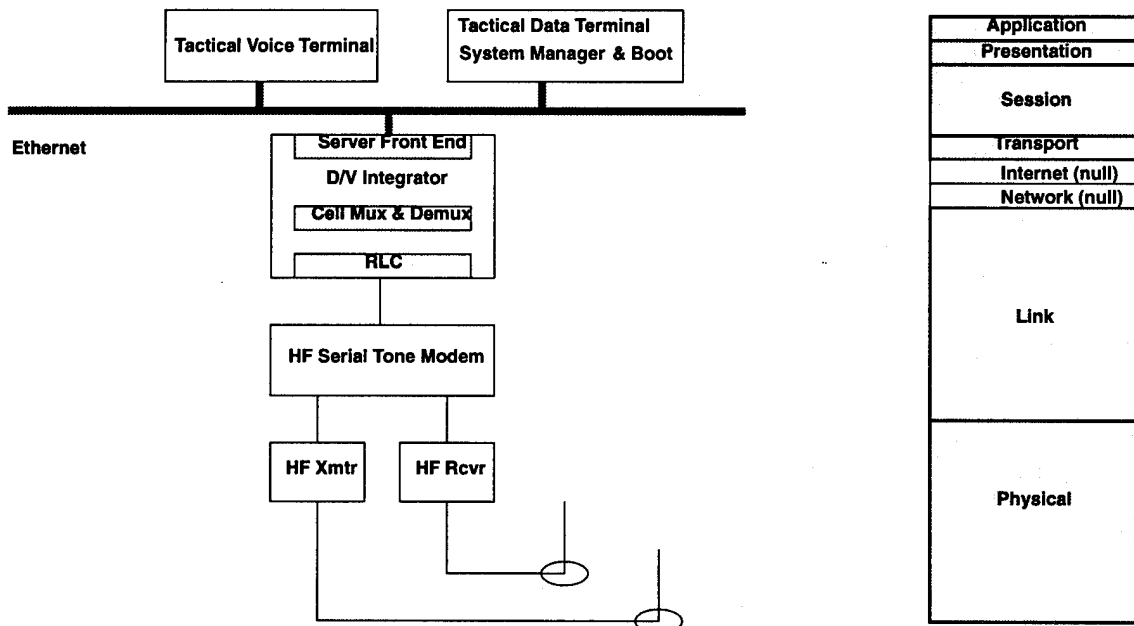


Figure 1. ATD Phase I system block diagram for a field demonstration node.

3.2 Phase II

The Phase II ATD node design shown in figure 2 shows an enhanced Application Layer with an IVOX Voice Terminal running on a SGI platform that supports a range of vocoding rates down to 600 bps. IVOX is an IP-based voice terminal application developed in-house at NRL. It combines low data rate voice compression algorithms with voice management protocols and a GUI. Also included is a version of a Joint Maritime Information Command System (JMCIS) that

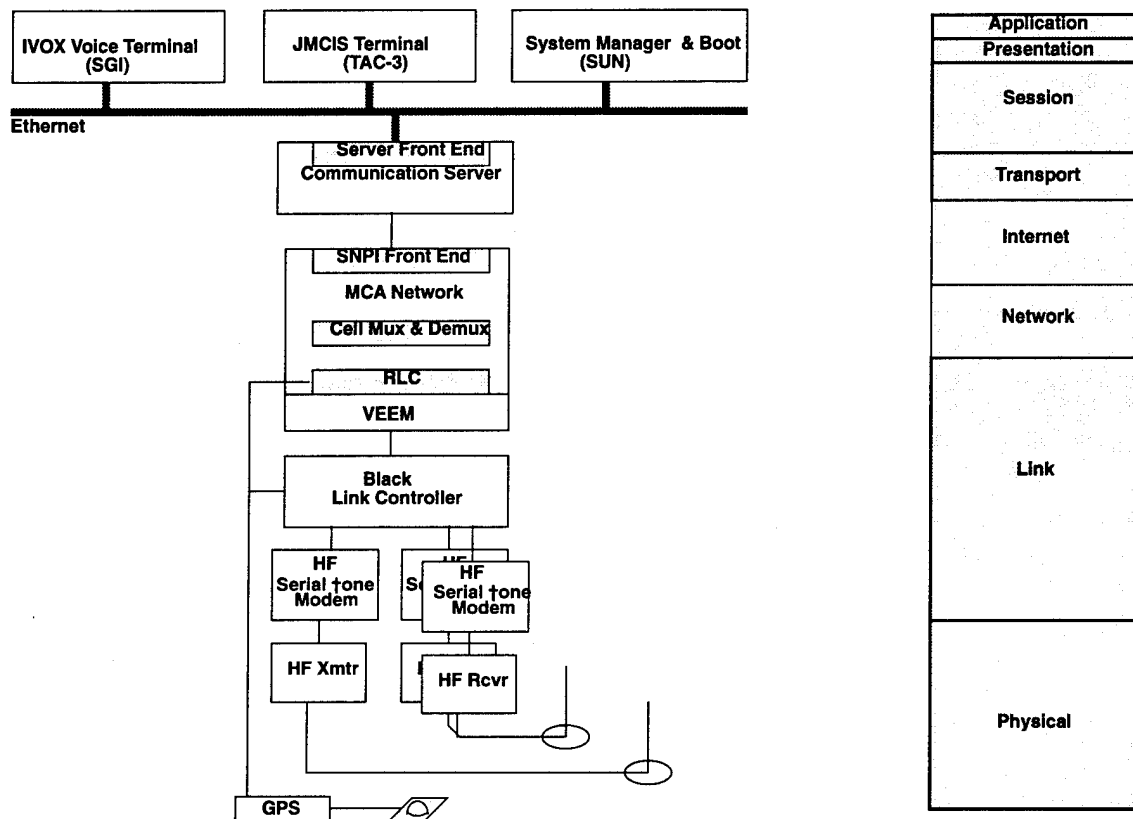


Figure 2. ATD Phase II system block diagram for a field demonstration node.

has been specially adapted for IP compatibility and will be further modified to take advantage of the API. The applications, such as IVOX, that use the API can negotiate special QOS parameters with the CS. Any of the Application Layer platforms can use MCA standard datagram service via the CS to support IP applications. The Phase II system uses link level, time-of-day based encryption (i.e., the VEEM - VME Embedded Encryption Module) to provide a COMSEC capability. The Black Link Controller (BLC) interfaces to the black side of the VEEM and controls several HF transmission strings.

3.3 Phase III

The Phase III design depicted in figure 3, shows the anticipated configuration of a Phase III demonstration node. The Phase III design includes an additional subnetwork in the form of a UHF SATCOM link over which the DVI multiplexing protocol can be run. Link encryption is provided by a KG-84. The CS can route IP packets through either subnetwork - MCA /HF or DVI/UHF.

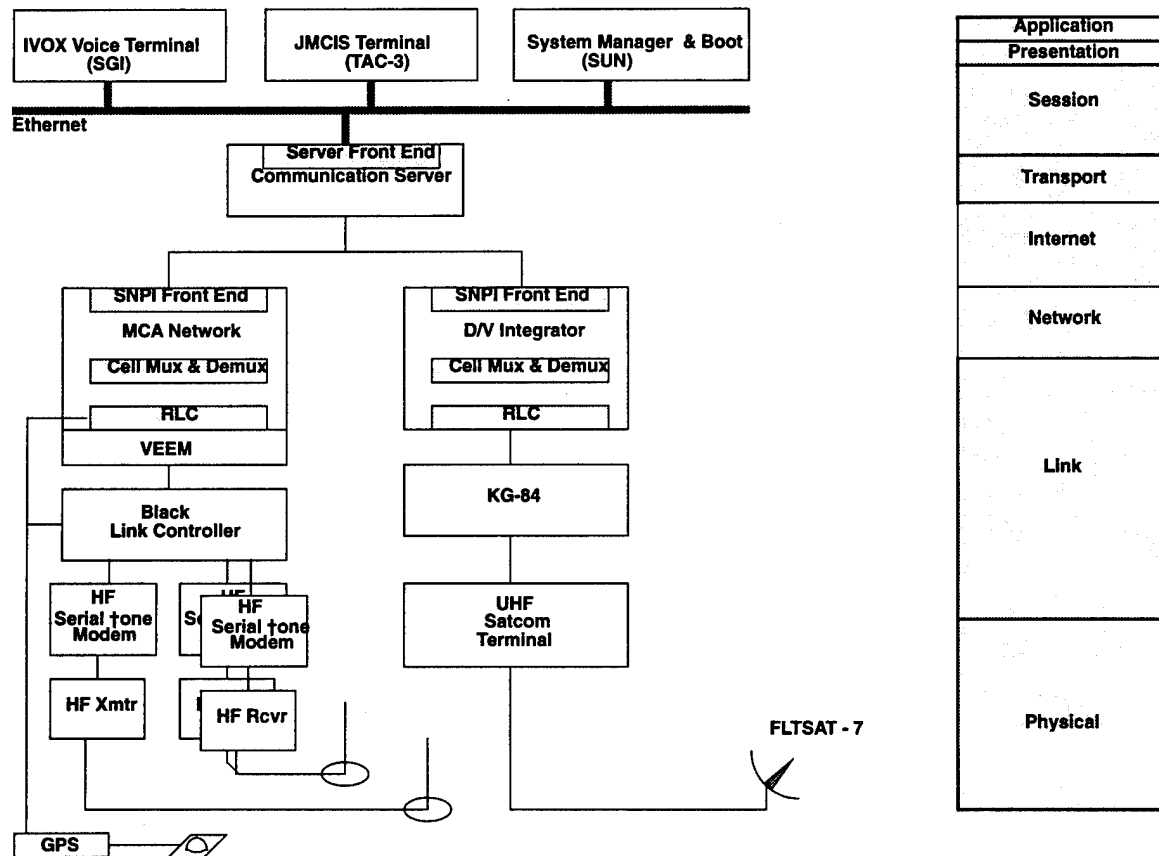


Figure 3. ATD Phase III system block diagram for a field demonstration node.

3.4 System Components

The following subsections briefly describe the system components shown in figures 1, 2, and 3.

3.4.1 Tactical Voice Terminal

The TVT implementation fielded for our December '93 demonstration was hosted on a 486 computer Workstation w/ ISA i860 dual processor card w/ 16 Mbytes of RAM. Because of problems maintaining adequate voice quality at 1200 bps, the ability to support only half duplex operation, and high latency within the TVT itself, the decision was made to pursue an alternate TVT design following the Phase I field demonstration.

NRL rapidly developed an alternative TVT in-house that was hosted on a Sun SPARCstation 10. This TVT supported full duplex voice operation at 1200 or 2400 bps, had low latency, and maintained consistently good voice quality at 1200 as well as 2400 bps provided the standard Sun microphones were replaced with high quality, directional microphones that adequately limited background noise. This version of a TVT was used successfully to demonstrate integrated voice

and data in several post Phase I laboratory demonstrations conducted at NRL in the Winter and Spring of 1994.

3.4.2 IVOX Terminal

The IVOX terminal is the result of more in-house development at NRL. We are currently hosting IVOX on Silicon Graphics (SGI) machines to gain the processing power necessary to run 600, 800, 1200, and 2400 bps vocoder algorithms in real time. Moreover, IVOX implements a GUI and a set of call management protocols that used the state machine diagrams presented in this report (figures 4 and 5) as a starting point for development.

3.4.3 Tactical Data Terminal

The tactical data terminal (TDT) demonstrated during the Phase I field demonstration and subsequent in-lab demonstrations consisted of several application programs written in-house that show a variety of capabilities. A partial list of these application programs follows:

- *atdpipe* - This application creates a text conversation capability. What a user types on his input screen is displayed remotely on another users output screen, and vice versa.
- *atd_jpeg* - This application supports reliable transfer of video images compressed in a jpeg format.
- *whiteboard* - This application supports a map redrawing and annotation capability whereby a user can update a remote user's screen and vice versa.
- *reliable_channel* - This application provides a reliable byte stream service.
- *test6* - This application, though it does not provide a service to the user, was a valuable driver for testing the robustness of the DVI's datagram handling and flow control mechanisms. It floods the DVI with datagrams for transmission until shut off by the DVI's flow control mechanism.

These Phase I data applications are being replaced with a more sophisticated suite of data applications for Phases II and III. The Phase I data applications were developed to demonstrate a concept of operation rather to provide a polished end product.

3.4.4 JMCIS Terminal

The Joint Maritime Command Information System (JMCIS) terminals, one per ATD node, are hosted on TAC-3 workstations. JMCIS maintains and displays track information. INRI, the company that developed JMCIS, has modified NRL's terminals to relay track information via a series of TCP/IP connections. Successful communication has been demonstrated among three JMCIS Terminals using TCP connections supported by the Phase II system operating in our laboratory environment. In this configuration, one JMCIS terminal acts as a relay for the other two, thus permitting all three terminals to present a consistent view of a developing scenario.

A desirable modification to this approach for Phase III would be to use multicast IP routing with resource reservation (RSVP) rather than multiple TCP connections. JMCIS would require some redesign to take advantage of this approach and it is unlikely that the D/V ATD will be able to fund further JMCIS development.

3.4.5 System Manager and Boot (SM)

The SM is hosted by a Sun SPARCstation. All other components, except for the application terminals, boot from the SM host. SM provides the means to control and monitor the ATD communication system. A detailed functional description of SM with some design information can be found in [3].

3.4.6 Data/Voice Integrator (DVI)

The DVI is implemented on a MVME167 single board computer hosted in a VME chassis and running the VxWorks 5.1.1 operating system. In the Phase I system, the DVI provided virtual circuit and datagram services directly to the client applications. In the Phase II and Phase III systems the CS component directly services client applications. Therefore, the DVI provides virtual circuit and datagram services indirectly to user applications via the CS in the current implementation.

3.4.7 MCA Network (MCA)

The MCA network also provides virtual circuit and datagram services indirectly to user applications via CS. It does so by relaying virtual circuit and datagram cell streams over a dynamically changing, multihop, HF groundwave network. MCA can work equally well over other RF media, e.g., UHF-LOS. In fact, MCA could be considered a better fit for UHF-LOS than HF-ELOS because of MCA's ability to support relaying with little loss in performance. Historically, the extended communication range offered by HF-ELOS has been used to compensate for the lack of relaying capability in Navy tactical networks. Running MCA over UHF-LOS would provide more bandwidth without sacrificing connectivity as long as each node in an MCA tactical network has at least one other node within communication range.

3.4.8 Red Link Controller (RLC)

Figures 1, 2, and 3 contain three different types of RLCs - the DVI/HF (Phase I) RLC, the MCA/HF (Phases II & III) RLC, and the DVI/UHF (Phase III) RLC. The DVI/HF (Phase I) RLC was hosted within the same processor card as the DVI and controlled a DataComm HF modem. For MCA, the RLC is hosted on a separate processor and provides the Red side data and control interface to the VEEM encryption module. Also, for MCA (Phases II & III) the RLC implements the required bypass controls to communicate with a Black Link Controller. The design for the DVI/UHF (Phase III) RLC is a modification of the DVI/HF (Phase I) RLC design to interface directly to a KG-84 (rather than the DataComm modem). This is the approach depicted in figure 3.

3.4.9 VME Embedded Encryption Module (VEEM)

The VEEM is a time-of-day (TOD) based cryptographic device hosted on a VME board. Command bypass and TOD synchronization functions are supported by the VEEM. The TOD synchronization capability eliminates the necessity to send lengthy synchronization preambles over the air. This significantly reduces synchronization overhead for packet radio networks, such as MCA, which must resynchronize their communication links frequently.

3.4.10 KG-84

KG-84s will provide transmission security over the Phase III UHF SATCOM link. The over-the-air synchronization preamble required by the KG-84 has a negligible impact on the full duplex, full period termination link that will support DVI transmissions for Phase III.

3.4.11 Black Link Controller (BLC)

The BLC controls HF modems and transmission hardware on behalf of the MCA network controller. MCA, via the RLC and VEEM bypass, issues commands to the BLC to retune receivers, resync the modems, and send and receive encrypted data. Each command given to the BLC indicates a time for execution of the command by the BLC. Thus, the BLC acts as a slave to the MCA network controller.

3.4.12 GPS

The Datum GPS receiver will provide a 1 ppms timing pulse to processor boards within both the RED and Black VME chassis. Timing information will be used to implement a time-of-day based synchronization protocol with the VEEM link encryption device.

3.4.13 UHF SATCOM Subsystem

A 25 kHz UHF SATCOM research channel is readily accessible to experimental users. A tentative plan for Phase III would place one terminal (one TD-1271 UHF SATCOM DAMA modem & two WSC-3 UHF SATCOM radios) at NRL and one at NRaD. NRaD would also provide the UHF SATCOM DAMA system controller and support for DAMA operation. DAMA, via the system controller, assigns slots of varying lengths and data rates to users, thus effectively partitioning a 25 kHz DAMA channel into subchannels of lesser bandwidths. Our intention is to acquire two non-adjacent DAMA slots, i.e., simplex subchannels, in order to support a full duplex UHF SATCOM link over which to run a pair of DVIs in full duplex mode. DAMA use has been mandated for MILSATCOM use by 1996.

3.4.14 HF Subsystem

The HF subsystem will use GAC MX-518 serial tone modems that support data rates <2700 bps. The MX-518 resides on a single VME board and provides a packet mode of operation featuring a relatively short sync preamble (80 ms). The HF receivers must be remotely tunable with a low latency retuning capability. The time required to retune a receiver adds directly to MCA overhead. Harris R-2368/URR receivers will be used for this purpose. MCA places less stringent requirements upon HF transmitters since the single transmitter of an MCA node does not require a remote control interface. Its frequency is not dynamically retuned as are the frequency settings of MCA receivers.

4.0 Tactical Voice Terminal Design

In order to achieve data/voice integration over a HF 2400 bps groundwave channel, a very-low-data-rate voice processor is required to encode/decode real-time speech at less than 2400 bps. The present IVOX implementation can perform vocoding at rates as low as 600 bps. Such low bandwidth capabilities for voice make it possible to support simultaneous data services over the remaining link capacity via an adaptive multiplexing scheme.

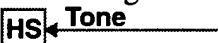

Another approach to further reducing the bandwidth requirements of voice is to detect silence gaps in the speech output stream and eliminate multiplexing the silent periods over the transmission media. The speech analysis output of the vocoder facilitates this approach since the amplitude parameter is a valid indicator of talk spurts and silence gaps. The DVI multiplexing scheme supports this approach directly since reserved but unused virtual circuit capacity is dynamically allocated to the datagram service.

The following subsections document the original TVT designs (which formed the basis for later IVOX development) for the Application/Session TVT functions and the TVT analog interface, i.e., handset. Also, the TVT requires an interface to the DVI. To properly specify the TVT/DVI interface it is first necessary to define the behavior of both the TVT and the DVI. The TVT/DVI interface is included as part of the more general DVI Server Interface and is covered in section 5.5 as part of the DVI design.

4.1 TVT System Behavior

Behavior of the TVT Application and Session functions is described in terms of expanded state machine diagrams as given in figures 4 and 5. Implementation of these state machines, one machine for half duplex and another for full duplex, requires the ability to sense the current state of the handset's on/off hook and, for half duplex, the push-to-talk (PTT) button. Also, the half duplex design, as it is presented in figure 4, requires that the DVI provide xon/xoff and service grant/deny control. The alternative to using xon/xoff would be to allow the TVT to send voice traffic to the DVI while the DVI is in a receiving state. In that case, the DVI would either have to buffer the voice traffic and then send it when the half duplex circuit turns around or discard voice traffic sent by the TVT while the DVI is in a receiving state. The TVT uses the Vc Svc Req command to request voice service from the DVI. The DVI responds by either granting the service request (Vo Svc Grnt) or denying it (Svc Not Grntd).

A walkthrough of figure 4 begins in the *Mode Selection* state with a user selecting half duplex TVT operation, thus causing the TVT to transition to the *Waiting* state. From the *Waiting* state, the TVT may transition to either the *Addr Selection* or the *Call Pending* states. An Off Hook event (i.e., the user removes the handset from its cradle) transitions the TVT to the *Addr Selection* state, permitting an address (phone number) to be entered. An Call Req event (i.e., incoming call) transitions the TVT to the *Call Pending* state.

Besides transitioning the TVT to the *Addr Selection* state, the Off Hook event has the added effect of sending a tone to the handset. This is depicted in figure 4 by an arrow pointing to a small box (). All such depictions are an expansion of the TVT's state machine diagram to show events that effect the states of other devices external to the TVT. These other devices are the handset, the DVI, and the remote TVT (). In some instances,

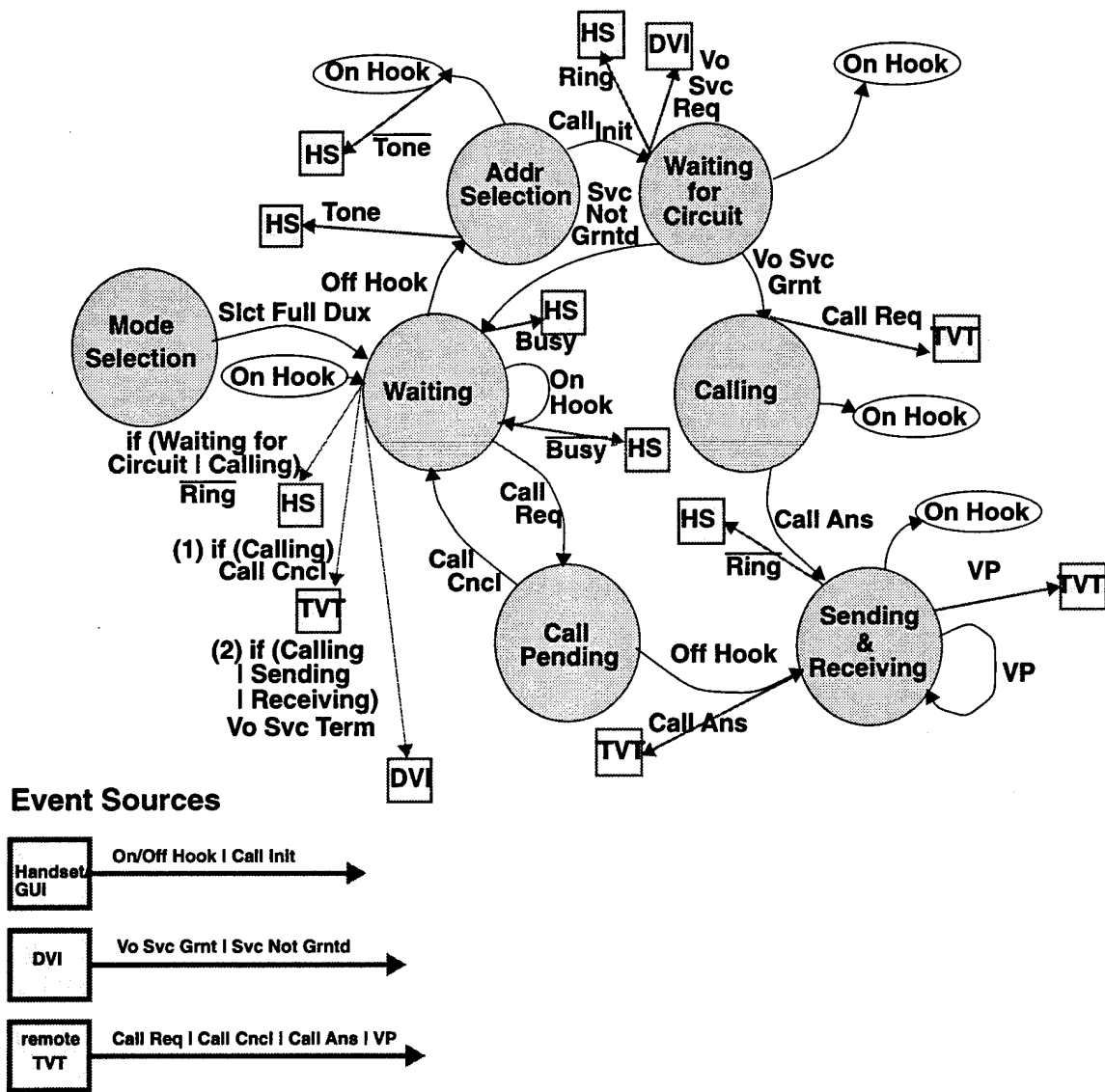


Figure 5. Expanded state machine diagram for TVT full duplex operation.

Table I: TVT State Machine Events

Event	Description
<u>Slt Half Dux</u>	This event results from a yet to be defined mode selection process that occurs while the TVT is in the Mode Selection state. The TVT should query the communication system to determine what modes (half or full duplex) and rates can be supported, report this information to the user, and then allow the user to select the desired mode and rate.

3

•

Table I: TVT State Machine Events

Event	Description
<u>On/Off Hook</u>	The standard meaning is assumed, i.e., the action of removing a handset from its cradle (off hook event) and the action of returning it to its cradle (on hook event). The on/off hook function can be emulated by the TVT's GUI in order to support handsets that do not have an on/off hook.
<u>PTT/$\overline{\text{PTT}}$</u>	PTT means that the push-to-talk button is asserted. $\overline{\text{PTT}}$ means that the push-to-talk button is deasserted. As with the on/off hook function, the PTT button can be emulated by the TVT's GUI, the TVT's keyboard, or the handset may have a PTT button.
<u>Call Init</u>	The Call Initiation event is sourced by the GUI (and/or keyboard or keypad) when an address selection has been made. The analogous event in the public phone system is entry of the final digit of a phone number. For the ATD system it might be preferable to have a separate button on the GUI or keyboard that initiates the call.
<u>Vo Svc Grnt</u>	In half duplex mode this is a message from the DVI indicating that the half duplex circuit is in local transmit mode so that a Call Req can now be issued by the TVT. In full duplex mode this is a message from the DVI indicating a full duplex circuit is available to support the call.
<u>Svc Not Grntd</u>	This is a message from the DVI indicating that it cannot support the voice service being requested at this time.
<u>Tx OK</u>	Tx OK completes a TVT/DVI handshake initiated when the TVT sends a Tx Hold command to the DVI. The DVI must be in the transmit state in order to return a Tx OK (see figure 8)
<u>Call Req/Cncl</u>	A Call Req is a message from a remote TVT placing a call. A Call Cncl message is received if the remote TVT decides to hang up the call before it is answered locally.
<u>Xon/Xoff</u>	Xon and xoff events are sent by the local DVI to indicate that the DVI is in the transmit mode (xon) or that the DVI is in receive mode (xoff).
<u>SOV/EOV</u>	SOV stands for <u>s</u> tart <u>o</u> f <u>v</u> oice traffic being received from a remote TVT. EOV stands for <u>e</u> nd <u>o</u> f <u>v</u> oice traffic being received from a remote TVT.
<u>VP</u>	VP represents either the sending or the receiving of a voice packet.
<u>Call Ans</u>	A Call Ans event occurs when a Call Ans message is received from a remote TVT, indicating that the pending Call Req has been answered by the remote TVT and that the remote user wishes to begin a conversation.

Table II: TVT State Machine States

State	Description
<u>Mode Selection</u>	Mode Selection is an initialization state in which the user decides whether to use half duplex or full duplex voice mode. When using MCA as a subnet, it may be necessary to do Addr Selection prior to Mode Selection since virtual circuits to one hop neighbors may support full duplex mode while virtual circuits requiring relays may support only half duplex.
<u>Waiting</u>	This is a quiescent state in which the TVT is waiting for voice management activity to occur. An On Hook event always returns the TVT to the Waiting state. The TVT can transition to two other states from the Waiting state: an Off Hook event transitions to the Addr Selection state or a Call Req event transitions to the Call Pending state.
<u>Addr Selec-tion</u>	The Address Selection state enables user entry of a callee address. The user may be assisted in this process by a GUI supported phone book.
<u>Waiting for Circuit</u>	The TVT waits for the DVI to respond to the Vo Svc Req. If a Vo Svc Grnt is received, the TVT transitions to the Calling state, or if the DVI returns a Svc Not Grntd, the TVT returns to the Waiting state. An internal ring (heard only via the Handset's receiver) is fed to the Handset.
<u>Calling</u>	The TVT is in the Calling state while a Call Req that it has issued to a peer TVT is pending. Either the peer TVT being called will respond with a Call Ans or the user who initiated the call will tire of waiting for an answer and hang up the phone, i.e., an On Hook event. When using the MCA subnet, it will be necessary to limit the time given to the callee to answer the call because of MCA time-outs used to reserve virtual circuit bandwidth that is not yet allocated.
<u>Call Pend-ing</u>	This is a state that is entered upon receiving a Call Req from a remote TVT and that produces an external ring (as opposed to a ring heard only via the Handset's receiver). The Call Pending state cannot be entered if an Off Hook condition exists. There are two ways to exit the Call Pending state: 1) the remote TVT hangs up the phone, thus sending a Call Cncl, or 2) an Off Hook event occurs, i.e., the user answers the phone.
<u>PTT Arbi-tration</u>	This is a state from which the PTT button is arbitrated. If the local PTT button is asserted, a Tx Hold is issued to the local DVI and the TVT transitions to the Tx Pending state. If the local DVI then returns a Tx OK, the TVT transitions to the Sending state. If an SOV event occurs, i.e., the DVI begins delivering voice traffic from a remote TVT, the local TVT transitions to the Receiving state. The net result is that when the TVT is in the PTT Arbitration state, either the local PTT button or the remote PTT button, via a SOV, will gain control of the TVT.

Table II: TVT State Machine States

State	Description
<u>Tx Pending</u>	When the TVT sends a Tx Hold to the DVI, it waits in the Tx Pending state for a DVI response. If the DVI can honor the Tx Hold command, i.e., the DVI is in the transmitting state, it returns a Tx OK and the TVT is enabled to transmit voice packets by transitioning to the Sending state. However, if the DVI is in the Receiving state, it can't service TVT transmissions and returns an Xoff to the TVT.
<u>Receiving</u>	This is a state in which the TVT is receiving voice traffic. The TVT cannot exit this state unless the EOv event occurs or the local user hangs up the phone.
<u>Sending</u>	This is a state in which the TVT is transmitting voice traffic. The TVT exits this state when the PTT button is deasserted or when an On Hook event occurs.
<u>Blocked</u>	This is a state in which the TVT is prevented from transmitting voice traffic by the DVI because the DVI is not in transmit mode.

4.2 TVT/TVT Messages

The TVT peer-to-peer protocol is implemented via a set of messages that pass voice data and control between TVTs. These messages are described in Table III and figure 7. Table III is a high level abstraction showing the data and control necessary to support a TVT peer-to-peer protocol. Figure 7 is a suggested TVT peer-to-peer message design. All messages have a one byte header and are an integral number of bytes in length. The SOV, EOv, and VP messages carry vocoded voice data. The remaining call management messages (Call Req, Call Ans, Call Cncl) use the payload to carry additional call management parameters. The sequence number in each message header facilitates detection of message losses. The virtual circuit service offered by the DVI will deliver messages in proper sequence, but messages may be lost or may contain bit errors.

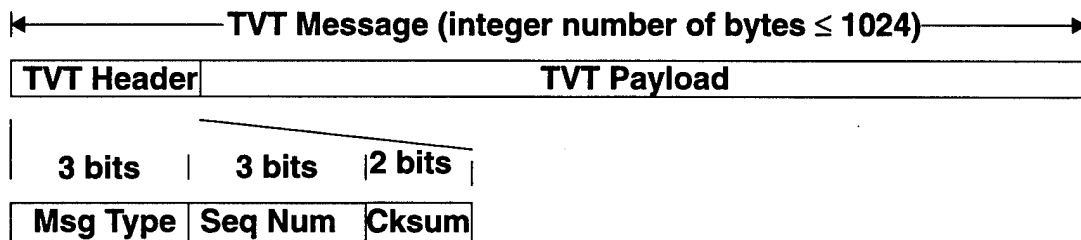


Figure 7. TVT message format

Table III: TVT End/End Protocol Messages

Type	Function	Parameters
Call Req	Requests a connection with a remote user	Half or full duplex; Vocoder algorithm/rate; Caller's id Callee's id
Call Ans	Remote user answers and connection is established	Caller's id Callee's id
Call Cncl	Connection request is canceled before connection is established	Caller's id Callee's id
SOV (Start of Voice)	First packet in a stream of voice packets	vocoded voice data
EOV (End of Voice)	Last packet in a stream of voice packets	vocoded voice data
VP (Voice Packet)	Packet of vocoded voice	vocoded voice data

5.0 Data/Voice Integrator Design

5.1 Overview

The DVI achieves data/voice integration by interspersing data and voice digital packets or *cells* at the origination node and transmitting the multiplexed stream of cells over a communication link. At the reception node, the voice and data streams are demultiplexed and appropriately reconstructed prior to presentation to the user. Here we develop a detailed DVI design for multiplexing the data and voice input streams into a single output stream at the source, feeding the stream to the RLC, and demultiplexing the stream at the destination. Actually, the services provided by the DVI are more generalized than its name suggests since the DVI integrates multiple virtual circuit and datagram services that can be used to support as many simultaneous users as the transmission bandwidth can handle.

5.2 Services Required from RLC

The DVI requires a digital transmission service from the layer beneath it. The characteristics of the service are as follows:

- The transmission service must handle loss of bit level synchronization. The DVI sends one cell at a time to the RLC for transmission and receives one cell at a time upon reception. Thus, the DVI pads the final cell of a transmission if it is not a full cell. Using fixed length blocks or "cells" provides a means to maintain synchronization with the DVI message stream in the presence of bit errors as long as the RLC provides a bit stream with the correct number of bits. Since it is possible for a UHF SATCOM channel to lose synchronization and thus gain or lose a bit, especially when running in a full period termination type of mode as it would for full duplex operation, the RLC must reestablish synchronization.
- The transmission service may be half or full duplex. However, if the service is half duplex, the RLC must support control of the transmit/receive switch.
- The transmission service should provide a nearly constant transmission rate for best DVI performance. The DVI can operate even if the transmission rate is not constant. However, the DVI may not be able to honor virtual circuit bandwidth reservations previously granted if the transmission rate suddenly decreases.
- The DVI can tolerate bit errors (not bit loss). It should perform well except in the presence of severe burst errors. In that circumstance, because of the small number of check bits used to detect cell and message header errors, undetected errors may occur. For a UHF SATCOM channel that provides $\sim 10^{-6}$ ber, the DVI will work efficiently and well. It should also work well over an HF ground wave channel at $\sim 10^{-3}$ ber. However, a burst error with a 50% ber will definitely generate some undetected errors. This type of condition can be detected by monitoring header error rates and is reported via DVI status messages; however, no corrective action is taken by the DVI in the current implementation.

5.3 Half Duplex Circuit Control

In the half duplex mode of operation, unless it is overridden by the PTT function of the TVT, the DVI is in control of link direction and will maintain a slot schedule by transmitting one

slot of data and then passing a token, i.e., an End-of-Transmission (EOT) message, to its DVI peer at the opposite end of the link. If there isn't enough data available to fill a slot, it will send a short slot. We propose 5 seconds as the duration of a slot¹ for DVI operation over UHF SATCOM.

When under control of the TVT, the DVI obeys *Rx On* and *Tx Hold* commands. The *Rx On* command switches the DVI from transmit to receive mode and appends an EOT message to the end of the transmission stream. At the other end of the link, the DVI that receives the EOT will switch from receive to transmit mode, thus reversing the link. On the other hand, the *Tx Hold* command prevents the DVI from switching to receive mode regardless of whether or not it has anything to transmit. If that is the case, the DVI will transmit *dummy cells* until traffic is available for transmission or until it receives an *Rx On* command from the TVT. This design is summarized in figure 8.

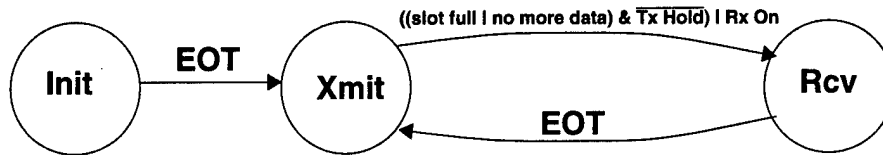


Figure 8. DVI state machine diagram.

5.4 DVI Messages

DVI messages must support several requirements. The first is the requirement to control the half duplex link by coordinating the transmit and receive states of the node pair that forms the link. This coordination is accomplished via an EOT message. Figure 10 presents two primary DVI message types: 1) an EOT message and 2) a Subchannel message. Both message types have the same length and exactly fit a cell. When a cell is received from the RLC, the DVI first determines if the cell is an EOT². If not, it is processed as a Subchannel message. Of course, the possibility exists that the EOT will be improperly determined. A solution for this problem is given in section 5.6.2. If a full duplex circuit is used, the EOT is unnecessary.

1. An important parameter in determining overall system efficiency is the slot length, which is defined as the number of bits transmitted in one direction over the half duplex circuit before the circuit is turned around. Two channel delays are required to turn the circuit around each time a slot is sent, yielding a circuit turnaround overhead, O_c as given by:

$$O_c = 2T_c / (T_s + 2T_c)$$

where T_c is the one way channel delay and T_s is the time required to transmit a slot. Thus, the way to keep turnaround overhead small is to make the slot transmission time large. On the other hand, long slot times reduce the responsiveness of the system by increasing the average time required to access the channel. A slot time of 5 seconds yields a 9.09% turnaround overhead given a one way channel delay of 250 ms and seems to be a good compromise between overhead and responsiveness.

2. The EOT message could also be a valid Subchannel message; however, the probability of generating a Subchannel message identical to an EOT is $1/2^n$, where n is the number of bits in an EOT message. On the other hand, an identical match will not be required to declare a message to be an EOT since a family of bit patterns will represent a valid EOT. Nonetheless, the family of valid EOT bit patterns is still quite small when compared to the complete set of 2^n patterns.

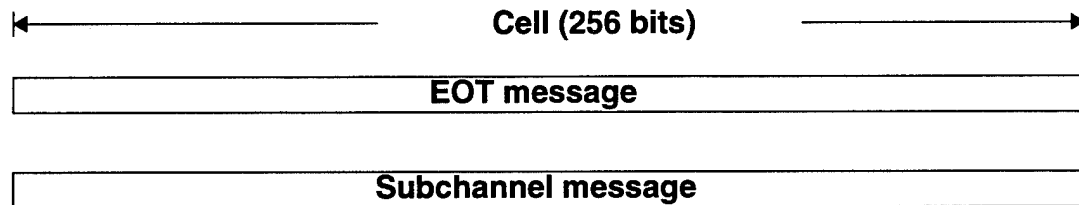


Figure 9. DVI peer-to-peer message format.

Secondly, DVI messages must support both a virtual circuit (VC) and a datagram (DG) service. These are the same services that are offered by the MCA network being developed for Phase II. The DVI and MCA have intentionally been designed to support identical services, as much as possible, to make the transition from DVI to MCA nearly transparent and to provide a consistent set of services at the subnetwork layer for the Phase III system. Before discussing the design details of VC and DG message formats, a discussion of VC and DG service requirements is in order. These requirements are presented in Table IV. An unreliable datagram service, ala UDP, is supported. ARQ could be added to provide a reliable datagram service, but this is not included in the Phase I DVI implementation. A message oriented virtual circuit service is implemented. (See Appendix 6.1 for a discussion of virtual circuit byte streams.) The VC message ser-

Table IV: DVI/MCA Service Requirements

Service Type	Service Requirements
Datagram	<ul style="list-style-type: none"> • no guaranteed throughput capacity • no guaranteed low latency - messages delayed beyond a time-out limit will be lost • duplicate messages removed (MCA only, NA for DVI) • messages may be lost • messages may contain errors • message order not preserved (MCA only, NA for DVI) • message boundaries preserved - read and write one message at a time (msg size \leq 8192 bytes)

Table IV: DVI/MCA Service Requirements

Service Type	Service Requirements
Virtual Circuit	<ul style="list-style-type: none"> • guaranteed throughput capacity • guaranteed low latency (\leq max value) - entire message must be received by DVI before the DVI delivers it • message boundaries preserved • message order preserved, but messages may be lost • messages may contain bit errors

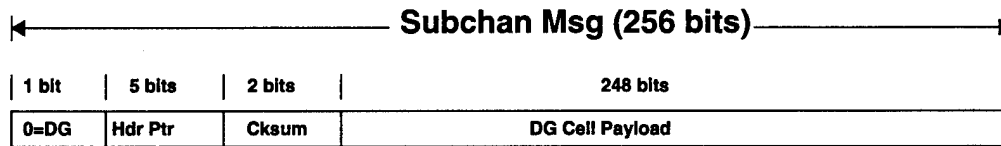
vice is much like the datagram service, except that the VC service expands the service capabilities to include throughput, latency, and message order guarantees.¹ Figure 10 defines formats for the VC and DG subchannel messages or *cells* and also defines formats for VC and DG messages that are carried in cell payloads, but are independent of cell payload boundaries. All cells have a fixed length of 32 bytes²; however, the number of bytes allocated to the cell header and payload differ with the cell type. DG cells use a one byte header and VC message (VCM) cells use two byte headers. The DVI also defines a set of VC control (VCC) cells which provide in-band control for setting up and terminating virtual circuits. The cell header parameters are defined as follows:

- *Cell Type (DG | VC)* - The first bit of a cell header determines the Cell Type, either Datagram (DG) or Virtual Circuit (VC).
- *Virtual Circuit Type (Msg | Cntrl)* - The second bit in VC cells gives additional type information by distinguishing the cell as a Virtual Circuit Message (VCM) cell or a Virtual Circuit Control (VCC) cell.
- *Cksum* - Each cell header contains a checksum that detects errors in the header (not the cell payload). If a cell header contains an error, then the cell type is not known and cannot be properly demultiplexed. Therefore, the cell is discarded and the contents of the cell payload is lost. VCC cells are an exception since the checksum in a VCC cell checks both the cell header and the cell payload.
- *Header Pointer (Hdr Ptr)* - DG and VCM cells carry messages via their cell payloads. They use the Header Pointer to locate the byte within the cell payload that contains the next message header. If a cell were never lost, the length parameter in the current message header (different than a cell header) would facilitate locating the next message header. However, cells will be lost. Thus, the Header Pointer guarantees that the next available message header can always be located.

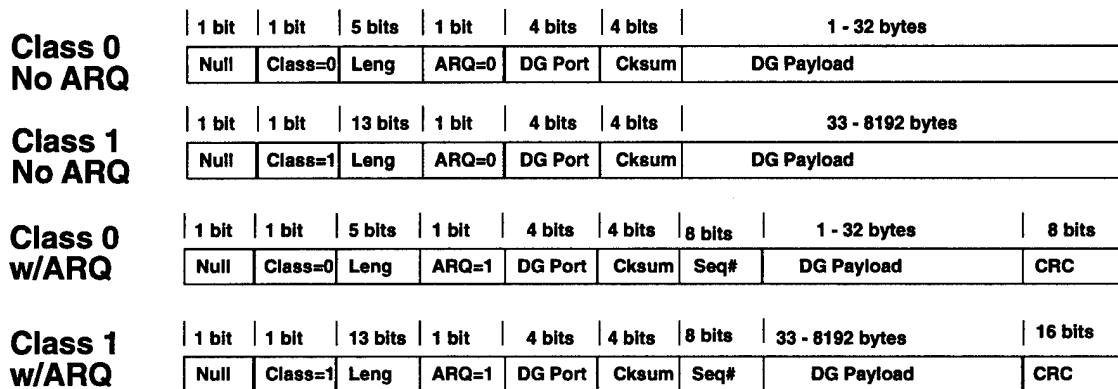
1. The reason that VC service can be characterized as similar to DG service but with more capabilities is because the DVI multiplexes services over a single link. The Phase II MCA network, on the other hand, requires additional architectural differences to support VC services when relaying is involved.

2. MCA uses a variation of the cell structure presented here; MCA cells are 36 bytes in length. Again, the differences between MCA and DVI are motivated by MCA's requirement to support relaying.

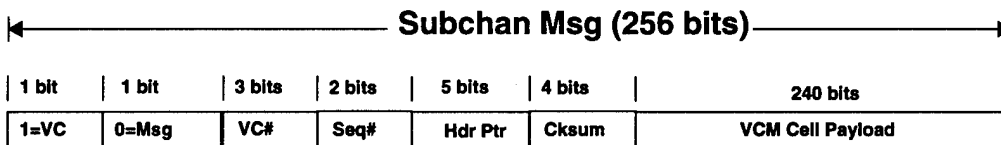
Datagram (DG) Cell



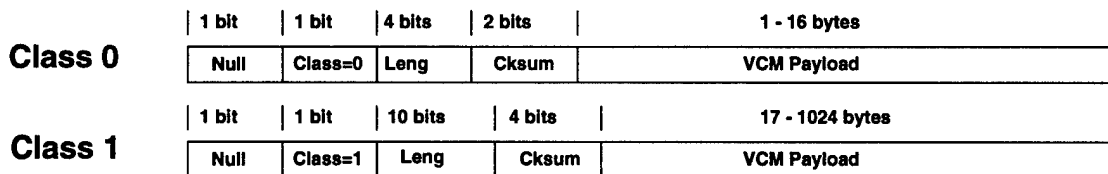
DG Messages



Virtual Circuit Message (VCM) Cell



VCM Messages



Virtual Circuit Control (VCC) Cell

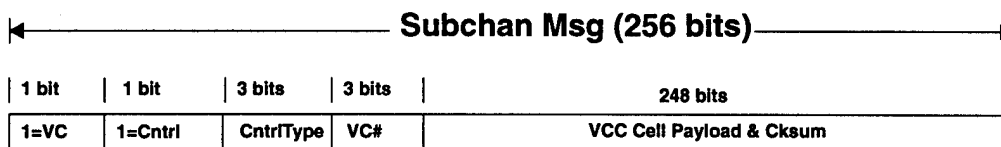


Figure 10. DVI datagram and virtual circuit cell and message formats.

- *Virtual Circuit Number (VC#)* - In order to support simultaneous virtual circuits, a Virtual Circuit Number is used to differentiate up to 8 virtual circuits. The VC# is carried by the cell header rather than by the VC message header contained in the cell payload because the DVI controls the allocation of VC capacities by controlling the number of VCM cells transmitted per unit time. This scheme is manageable because all cells are the same length. If, on the other hand, we attempted the multiplexing of several virtual circuits at the VC message level instead of the cell level, control of capacity allocated to each virtual circuit would be more difficult because the potential variations in VC message sizes are controlled by the users of VCM services rather than the DVI.
- *CntrlType* - Virtual circuit control cells are further categorized by the 3 bit CntrlType. Table V describes the various VCC cells.

Table V: Types of Virtual Circuit Control Cells

CntrlType	Cell Name	Function
0	vcSetup	This cell sends virtual circuit setup request parameters from DVI initiating the call to DVI receiving the call.
1	vcSetupCont	This cell sends additional setup parameters provided by DVI client originating the call.
2	vcSetupRply	This cell is used to reply to the virtual circuit set up request, indicating acceptance or denial of the request.
3	vcTerminate	This cell is used by either DVI to terminate a virtual circuit.
4	vxRxOn	This cell is not sent over-the-air. It is used when operating in half duplex mode as a flag in a virtual circuit's transmission stream to switch from transmit to receive. This insures that the DVI commands the RLC to switch the transceiver when the last cell queued for transmission has been sent.

The DG cell format in figure 10 is followed by formats for four types of DG messages. The following parameters are supported by the DG message header:

- *Null* - Every valid message begins with a *Null* (0 bit) to distinguish it from a dummy message.
- *Class* - The Class indicator supports a short (Class 0) and a long (Class 1) datagram. Class 1 requires an additional byte of length information in the DG message header.
- *Leng* - The Length parameter specifies the length of the DG message payload (not the length of the datagram itself) and is given as the payload length minus 1.
- *ARQ* - The ARQ bit specifies whether or not a Seq# is added to the header and a Cksum is added to check the DG payload. The ARQ bit anticipates eventual implementation of an

ARQ scheme.

- *DG Port* - The DG Port number provides the capability to support up to 16 datagram service ports within the DVI. If ARQ is implemented, Port 0 will be used to handle acknowledgments, leaving 15 ports to service DVI users.
- *Cksum* - The checksum detects errors in the DG message header. If the header contains an error, the entire datagram will be discarded.
- *Seq#* - Sequence numbers will be required by a message level ARQ scheme.
- *CRC* - A cyclical redundancy checksum is used for the DG payload in order to implement an ARQ scheme.

VC cells carry two kinds of VC messages, as shown in figure 10. The Class 0 VC message uses a one byte header while the Class 1 message uses a two byte header. VC messages are similar to DG messages, except that VC messages do not use an ARQ bit or a DG port number. Virtual circuits cannot meet requirements for latency and capacity if they also attempt to guarantee reliability via an ARQ scheme. Therefore, the ARQ bit is not needed. Also, since VC cell headers specify a VC#, a port number is not needed in the VC message header.

5.5 DVI Service Interface

As described in Table IV, the DVI supports two basic types of service, a datagram and a virtual circuit service. In this section we describe the Phase I interface design by which these basic services and a supplemental service for half duplex voice are accessed. For Phases II and III this design has been slightly modified and extended. It is called the Subnetwork Provider Interface (SNPI) and is documented in [3]. A brief overview of the SNPI is provided in section 6.2.

5.5.1 DVI Server Access Via TCP

Clients wishing to access DVI services must first establish a connection to the DVI via a TCP socket. The actions of the client are as follows:

- create a stream socket
- connect to the server via a call to *connect()*

The DVI in its roll as server must perform the following actions:

- create a stream socket
- bind the socket to a well known port
- create a connection request queue
- listen for a connection
- accept a connection and spawn a task to service the connection

Once the connection is established, TCP provides a reliable, full duplex, byte stream service. Later subsections (5.5.3 through 5.5.9) describe messages that are passed back and forth between client (i.e., the DVI user) and server (i.e., the DVI) via the TCP connection to manage the DVI service. Since TCP is a stream service, it transports information in units of bytes, not messages. Thus, both client and server need to support a protocol to identify message boundaries. The protocol used in the DVI client/server message set that follows is to give the message length at the same location in every message. Then, the length can be decoded to locate the end of the message

and, subsequently, the beginning of the following message and its length. Before discussing client/server messages, section 5.5.2 gives additional discussion related the use of TCP connections to provide client access to the DVI server.

5.5.2 DVI as a “TCP Bridge”

A pair of DVIs communicating with each other over a link create a “TCP bridge” for the client processes that they are serving. Figure 11 illustrates the “TCP bridge” concept. In order to

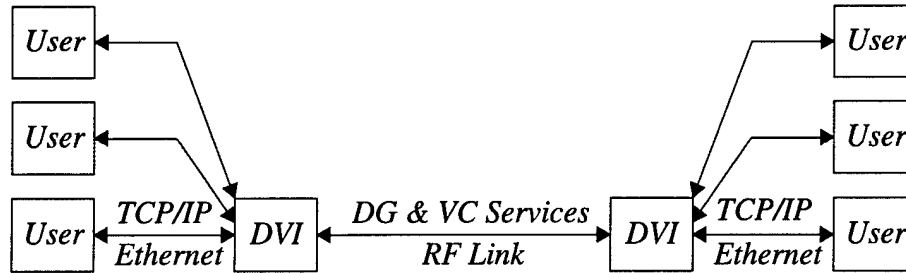


Figure 11. Phase I configuration using a pair of DVIs acting as a “TCP bridge.”

communicate with a peer user, a local user establishes a TCP connection with the local DVI. The local and remote DVIs use their own cell multiplexing technique to transport end-to-end user data. At the remote end a TCP connection is again used to deliver the data to the remote user. This is the design implemented for Phase I. Phases II and III modify this approach so that a user (i.e., the Communication Server) does not use the MCA network or the DVI link as a TCP bridge. Rather, the DVI and MCA datagram and virtual circuit services are used to transport TCP/IP and UDP/IP packets across RF links and networks. Thus, TCP (and UDP) connections will be formed not between a user and a DVI, but directly between users. This approach is known as “tunneling” and is illustrated in figure 12. In this figure, the Communication Server (CS) acts as a gateway/router. IP packets that must be routed over RF links or networks are packed into DG or VC sub-network layer messages as appropriate and sent via DG or VC services.

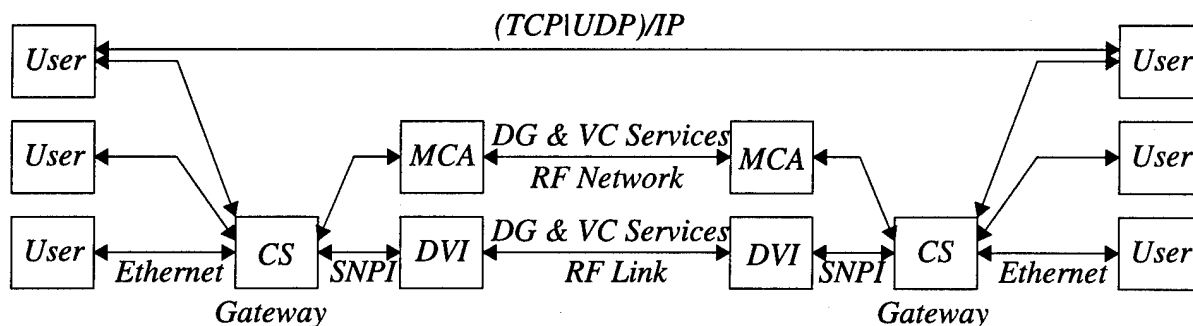


Figure 12. Phase III configuration showing a pair of CS gateways “tunneling” IP via DVI and MCA transport services.

5.5.3 Client Registration

Upon establishing a TCP connection with the DVI server, a client must register for service by sending its user id. The purpose in providing a registration request in addition to a service request (see section 5.5.4) is to enable the DVI to deliver datagrams or VC service notifications that it has received for a passive client, i.e., a client that is not currently engaged in using a service. In essence, client registration makes the client's presence known to the DVI and enables the DVI to associate a particular client with a particular TCP socket. Client registration is handled by the messages given in Table VII.

Table VI: Messages for Client Registration

Message Name	Sent By	Parameter Names & Values
rgsReq	Client	• User id: <u>ARQ</u> , <u>TT1</u> , <u>TT2</u> ,..., <u>TT15</u>
rgsOK	Server	None
rgsFail	Server	• Reason: <u>BadUid</u> <u>UidInUse</u> <u>TooManyClnts</u>

5.5.4 Opening and Closing a DVI Service

Opening and closing DVI services are handled by the messages given in Table VII. The *dgSvcReq* specifies the type of datagram service (polled or unpolled) being requested. The *vcSvcReq* specifies the type of VC service being requested as well as additional parameters that specify precedence, rate, average VC message size (to aid the DVI in determining the amount of actual communication capacity required to support the rate), and a destination address. The precedence and destination address are supplied independently for each datagram that is sent, but only once when a virtual circuit is set up. The notification messages, *svcNtfy* and *svcTermNtfy*, are delivered by the remote DVI to a remote user callee to indicate that virtual circuit service has been set up or terminated by the caller. The *svcTerm* message is used by a caller or a callee to terminate VC service. Finally, the *svcGrntd* and *svcNotGrntd* messages are sent by the DVI server to the client making a service request to indicate the status of the request. If the DVI honors the service request, a *svcGrntd* message is returned to the client along with a Service Id. The Service Id must be included by the client in all subsequent commands related to that service. The *svcNotGrntd* message indicates to the client that service was not granted. The reason that service was not granted is given in the first return parameter. If the *svcNotGrntd* message is returned in response to a *vcSvcReq*, two additional parameters are supplied - the maximum capacity available to support the VC service at the requested precedence and the minimum precedence level, if it exists, that could be used to acquire the requested capacity. With this information, the client may negotiate with the DVI by formulating a new service request that the DVI can honor.

5.5.5 Datagram Service

Use of the message set given in Table VII for Datagram service presupposes that a client has successfully obtained the service from the DVI via a *dgSvcReq* / *svcGrntd* message exchange. Once datagram service is established, the client may send either the *dgSnd* message to the DVI, in

Table VII: Messages for Opening and Closing DVI Services

Message Name	Sent By	Parameter Names & Values
dgSvcReq	Client	<ul style="list-style-type: none"> Service Mode: <u>Polled</u> <u>Unpolled</u>
vcSvcReq	Client	<ul style="list-style-type: none"> Service Type: <u>VCsmplx</u> <u>VCswsmplx</u> <u>VCdplx</u> <u>VCswdplx</u> Precedence: <u>Routine</u> <u>Immediate</u> <u>Priority</u> <u>Flash</u> <u>FlashOverride</u> Rate: requested capacity in bps Average VC message size in bytes client expects to use Destination Address (User Id + Node Id)
svcNtfy	Server	<ul style="list-style-type: none"> Service Id Service Type Precedence Rate: capacity in bps that is reserved for this service Average VC message size in bytes that is the basis of the capacity reservation Source Address (user & node that initiated service)
svcTerm	Client	<ul style="list-style-type: none"> Service Id
svcTerm-Ntfy	Server	<ul style="list-style-type: none"> Service Id
svcGrntd	Server	<ul style="list-style-type: none"> Service Id
svcNtGrntd	Server	<ul style="list-style-type: none"> Reason: <u>BadSvcType</u> <u>RateTooHigh</u> <u>NotRgstrd</u> <u>TooManySvcs</u> <u>TooManyVcns</u> <u>CalleeBadSvcType</u> <u>CalleeRateTooHigh</u> <u>CalleeNotRgstrd</u> <u>CalleeTooManySvcs</u> <u>CalleeTooManyVcns</u> <u>Other</u> Maximum capacity DVI can provide at requested precedence level Minimum precedence required to support requested rate

order transmit datagrams, or the *svcTerm* message, in order to close the datagram service. The client may also close the TCP connection to terminate a service. The DVI server may send three types of messages to the client while datagram service is in operation. The *dgRcv* message is used to deliver datagrams from a remote peer to the local datagram client. The *dgPoll* message is used only when the client has requested a polled mode for sending datagrams. The *dgStatus* message is returned to the client after every *dgSnd* to indicate the status of the polled and unpolled services.

Table VIII: Messages for Managing DVI Datagram Services

Message Name	Sent By	Parameter Names & Values
dgSnd	Client	<ul style="list-style-type: none"> • Service Id • Destination Address (Node Id + User Id) • Length of data in bytes • 1 - 8192 bytes of data • Precedence of data: default Routine
dgRcv	Server	<ul style="list-style-type: none"> • Length of data in bytes • 1 - 8192 bytes of data
dgPoll	Server	<ul style="list-style-type: none"> • Service Id
dgStatus	Server	<ul style="list-style-type: none"> • Status of most recent dgSnd: <u>Ok</u> <u>BadSvcId</u> <u>MsgTooLarge</u> <u>MsgQueueFull</u> <u>BadParameters</u> <u>MsgBufferLocked</u> (polled service) <u>SndFailure</u>

5.5.6 Virtual Circuit Service

Since virtual circuit service is message oriented, it is handled in a fashion similar to the datagram service, as shown in Table VII. The differences are that virtual circuit service does not support a polling mechanism and that a virtual circuit enforces the allocated circuit capacity. The DVI currently implements half-duplex (VCswsmplx), full-duplex (VCdplx), and simplex (VCsmplx) virtual circuit services. The TVT designed for Phase I does not support the call management protocols described in section 4.0. Rather, full-duplex service is obtained by manually starting voice source and sink processes on a pair of TVTs which then establish a pair of simplex virtual circuits, one in each direction. Subsequent versions of the TVT will implement call management protocols.

5.5.7 Half Duplex Voice Service

Half duplex voice service is handled as a special case because additional TVT/DVI control is needed to reverse the direction of the half duplex link. Half duplex voice uses the message

Table IX: Messages for Managing DVI Virtual Circuit Services

Message Name	Sent By	Parameter Names & Values
vcSnd	Client	<ul style="list-style-type: none">• Service Id• Length of data in bytes• 1 - 1024 bytes of data
vcRcv	Server	<ul style="list-style-type: none">• Length of data in bytes• 1 - 1024 bytes of data
vcStatus	Server	<ul style="list-style-type: none">• VcSnd: <u>Ok</u> <u>BadSvcId</u> <u>MsgTooLarge</u> <u>CapacityExcd</u> <u>UrateExcd</u> <u>InRcvngState</u> (half-duplex) <u>Preempted</u> <u>SndFailure</u>

set provided above in Table VII for sending a receiving data, i.e., TVT/TVT messages. It uses the message set given in Table VII for half duplex voice control of the DVI.

Table X: Messages for Half Duplex Voice Control of a DVI

Message Name	Sent By	Parameter Names & Values
hdvRxOn	Client	<ul style="list-style-type: none">• Service Id
hdvTxHold	Client	<ul style="list-style-type: none">• Service Id
hdvXoff	Server	<ul style="list-style-type: none">• Service Id
hdvXon	Server	<ul style="list-style-type: none">• Service Id
hdvTxOK	Server	<ul style="list-style-type: none">• Service Id

5.5.8 Flush

A *dviflush* message is sent by a client to the DVI server to force transmission of the last datagram or virtual circuit message. Since the current DVI implementation performs automatic flushing, the *dviflush* message is not needed. However, the DVI could be easily reconfigured to limit automatic flushing in order to conserve bandwidth and the *dviflush* command would then become useful.

5.5.9 DVI Status

There are two ways to obtain DVI status information. The first is via the vxWorks rlogin shell and the second is by using a client process that sends *statusReq* and *statusReset* messages to the DVI and receives *statusRply*, *statusDgSvc* and *statusVcSvc* messages in return (see Table VII). The *dviHelp* command prints the message shown in figure 13 and lists other commands that may

Table XI: Flush Message

Message Name	Sent By	Parameter Names & Values
dviflush	Client	• Service Id

be used to print DVI statistical variables, reset statistical variables to zero, or turn message tracing on and off. Two types of message tracing are available, one that displays messages traversing the

dviHelp	print this msg	<i>Time since startup or last reset</i>
pt	print elapsed time & measured transmission rate	<i>Current transmission rate in bps</i>
pdvi	print dvi stats	<i>Print all stats</i>
pcell	print cell stats only	
pm	print msg stats only	
pu	print user stats only	
ps	print service stats only	
pq	print vc cellQ stats only	
rdvi	reset dvi stats	
rcell	reset cell stats only	
rmsg	reset msg stats only	
ru	reset user stats only (not implemented)	
rs	reset service stats only (not implemented)	
rq	reset vc cellQ stats only	
t1on	turn client/server msg tracing on	
t1off	turn client/server msg tracing off	
t2on	turn RLC msg tracing on	
t2off	turn RLC msg tracing off	

Figure 13. Commands available from VxWorks rlogin shell for displaying/resetting statistical variables and turning tracing on/off.

client/server interface and another that displays messages traversing the DVI/RLC interface. The print commands give the following information about DVI performance:

- *pcell*
 - ◇ xmtCellCnt & rcvCellCnt - total number of cells sent and received (including dummy cells)
 - ◇ xmtDmyCellCnt & rcvDmyCellCnt - total number of dummy cells sent and received
 - ◇ dgXmtCellCnt & dgRcvXmtCellCnt - total number of DG cells sent and received
 - ◇ vc[0-7]XmtCellCnt & vc[0-7]RcvCellCnt - total number of VC cells sent and received for a given virtual circuit number

- ◇ cellHdrErrCnt - number of cell header errors detected
- ◇ rlcResetCnt - number of times RLC was reset because of total garbage being received
- *pm*
 - ◇ dgXmtMsgCnt & dgRcvMsgCnt - total number of datagrams sent and received
 - ◇ vcXmtMsgCnt[0-7] & vcRcvMsgCnt[0-7] - total number of virtual circuit messages sent and received for each virtual circuit number
 - ◇ dgMsgHdrErrCnt - total number of errors detected in DG message headers
 - ◇ vcMsgHdrErrCnt - total number of errors detected in VC message headers
- *pu* - following information provided for each registered user
 - ◇ USER - integer value of the uid for each user registered with the DVI
 - ◇ SOCKET - value of the TCP socket file descriptor that connects the DVI to the user
 - ◇ SIDs - list of service identifiers for services granted to each user
- *ps* - following information provided for each user/service
 - ◇ uid
 - ◇ service type
 - ◇ for DG service - polled or unpolled
 - ◇ for VC service - priority, VCN, requested rate and average message size, current sample of user's input rate integrated over sliding window size (default = 5s), maximum and average user input rate, capacity allocated for the VC, actual capacity used by VC (may exceed allocated capacity when VC cells are flushed)
- *pq*
 - ◇ VC cell queue statistics for each VCN - average queue size, maximum queue size, current queue size, VC cell pool size
 - ◇ DG cell queue statistics - average queue size, maximum queue size, current queue size, DG cell pool size

Table XII: Messages for Requesting and Sending DVI Status

Message Name	Sent By	Parameter Names & Values
statusReq	Client	<ul style="list-style-type: none"> • StatReqType: <u>UnpollOpr</u> (client requests unpolled operation - DVI will periodically send stats to client) <u>StopUnpollOpr</u> (client stops unpolled operation) <u>PollAll</u> (Poll for all stats: DVI, VC, & DG) <u>PollDvi</u> (Poll for DVI stats) <u>PollDg</u> (Poll for DG service stats) <u>PollVc</u> (Poll for VC service stats)

Table XII: Messages for Requesting and Sending DVI Status

Message Name	Sent By	Parameter Names & Values
statusRply	Server	<ul style="list-style-type: none"> • struct statusRprt
statusDgSvc	Server	<ul style="list-style-type: none"> • struct statusRprtDgSvc
statusVcSvc	Server	<ul style="list-style-type: none"> • struct statusRprtVcSvc
statusReset	Client	<ul style="list-style-type: none"> • StatReqType: <u>ResetAll</u> (Reset for all stats: DVI, VC, & DG) <u>ResetDvi</u> (Reset for DVI stats) <u>ResetSvc</u> (Reset for service stats)

5.6 DVI Operation

The DVI design specifies two phases of operation - an initialization phase and an operational phase. In full duplex mode, the Initialization Phase consists of creating and binding a TCP socket as well as spawning the Multiplexer and Demultiplexer tasks. In half duplex mode the same initialization occurs plus initialization for the half duplex link. Figure 14 gives a flow chart of DVI operation as it relates specifically to initializing and maintaining the link in half duplex mode. The details of the Operational Phase will be further described in succeeding figures.

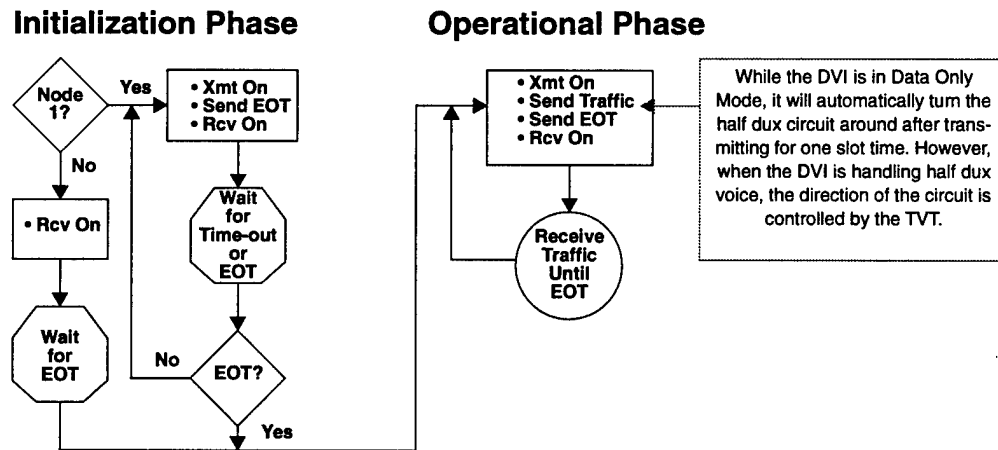


Figure 14. Flow chart of DVI half duplex operation.

5.6.1 Initialization Phase

Each DVI has a unique node number assigned to it. Node 1 is the first node to transmit during the initialization phase. It transmits an EOT and waits for a response from its DVI peer at the opposite end of the link. If it doesn't hear an EOT in response, it continues to repeat this procedure until it does. Node 2 doesn't transmit until it hears an EOT from Node 1. When it does it

can send its first slot of data, ending with an EOT. Node 1 then sends its first slot of data, terminated with an EOT, and so on.

5.6.2 Misinterpreted EOTs

When an EOT is misinterpreted, the half duplex link will be lost. If an EOT is sent and the receiver misses it, the sender will switch to receive mode while the receiver remains in receive mode, leaving both nodes in receive mode. On the other hand, if a Subchannel message is falsely interpreted as an EOT message, the receiving node will switch to transmit mode while the transmitting node remains in transmit mode. Thus, both nodes are transmitting; neither are receiving. In the first case, i.e., both nodes receiving, each node will return to the Initialization Phase after waiting for a time-out period and not receiving a transmission. The second case, i.e., both nodes transmitting, eventually evolves to the first case if we assume that a node, having missed the modem synchronization preamble, cannot get into proper byte sync with the incoming data stream even if it could receive it. If this assumption is false, then one node switches to receiving mode, gets in step with the received byte stream, and we again have a link. Thus, half duplex link synchronization can be recovered after an EOT failure.

5.6.3 Operational Phase (Cell Packing, Multiplexing, & Data/Voice Integration)

The functions of the Operational Phase are the following:

- manage the service interface provided to DVI clients
- manage datagram and virtual circuit services
- pack and multiplex the stream of cells to be transmitted
- demultiplex and unpack cells that have been received
- deliver DG and VC messages to proper recipients
- manage half duplex circuit switching (if in half duplex mode)
- manage the interface to the RLC

The functions just listed, with the exception the cell packing and multiplexing schemes, are discussed in other sections of the documentation. Therefore, cell packing and multiplexing are described here and it is here that data/voice (i.e., datagram and virtual circuit) integration actually takes place.

Voice and data are multiplexed onto the DVI's output stream by interleaving VC and DG cells. By using 3 bits to specify a VC#, up to eight unique VC cell streams can be multiplexed over a DVI link. Since DG cells do not specify a DG#, the DVI supports only one DG cell stream. However, the DG messages that are packed within the DG cell stream contain a 4 bit DVI Port number which allows multiplexing of up to 16 DG subchannels over the one DG cell stream. Figure 15 illustrates the multiplexing of individual, homogeneous cell streams to form a heterogeneous output stream. Picturing VC and DG cells as streams is not meant to imply anything about buffering or queuing of cells at the source. However, the concept of individual streams being interleaved into a multiplexed output stream is useful in understanding how cell payloads are packed, as shown in Figure 16.

Figure 16 applies equally well to the creation of VC message (VCM) or DG cells. In either case, messages sent by a client are first encapsulated as a VCM or a DG by attaching the

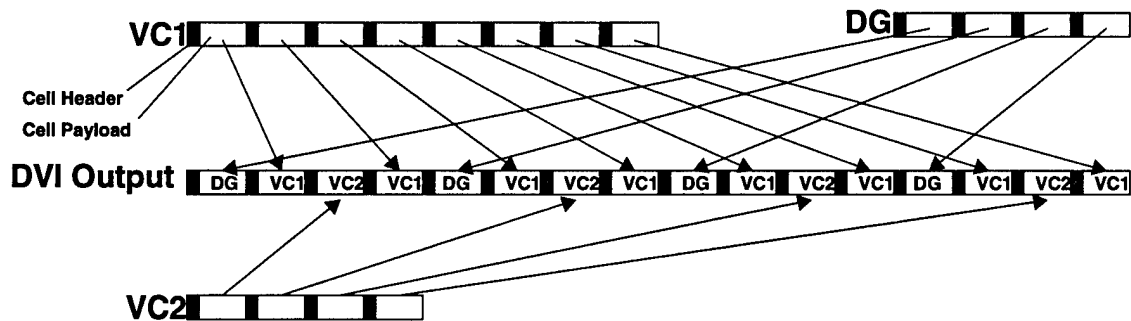


Figure 15. Conceptual view of multiplexed cell streams to form a DVI output stream.

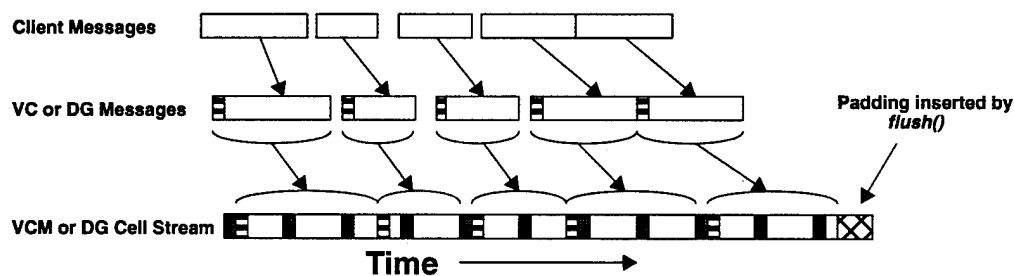


Figure 16. Packing of Virtual Circuit Message and/or Datagram Cells

appropriate VCM or DG header. Then the DVI contiguously packs the VCMs or DGs into cell payloads. When the DVI is heavily loaded, a cell payload is completely filled before it is placed into a FIFO queue (one queue per individual cell stream) to await insertion into the output stream by the multiplexer. On the other hand, if DVI traffic loading is light, the DVI will use the excess capacity to automatically flush a partially filled cell in order to expedite the cell's transmission when additional data is not yet available to fully pack the cell. This process is further elaborated in figure 17.

The Multiplexer Task maintains a FIFO queue for each cell stream, i.e., one for the DG cell stream and one for each of the eight possible VCM cell streams. Based on the availability of cells in each FIFO queue, the relative priority of each stream, and, in the case of VCM streams, the capacity requirements, the Multiplexer Task determines the order in which cells are dequeued and inserted into the output stream. The protocol that controls cell multiplexing is shown in figure 18.

The cell multiplexing control loop operates on the service ports that have been created and placed in the Service Port Queue. One service port is created to handle all datagrams, one service port is created to handle all DVI peer-to-peer commands, and service ports are dynamically created to handle virtual circuits.

Every service port has a FIFO cell queue and a capacity bound, i.e., a limit that is used to control the transmission capacity allocated to the service port. The DG Service Port's capacity bound is always set to zero, meaning that no capacity is reserved for datagram service. The CMD Service Port's capacity bound is set to infinity, meaning that DVI peer-to-peer commands will get

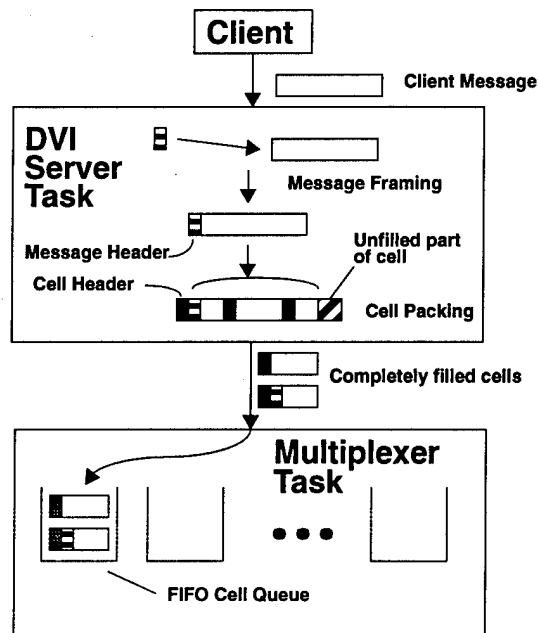


Figure 17. Cell creation and queuing.

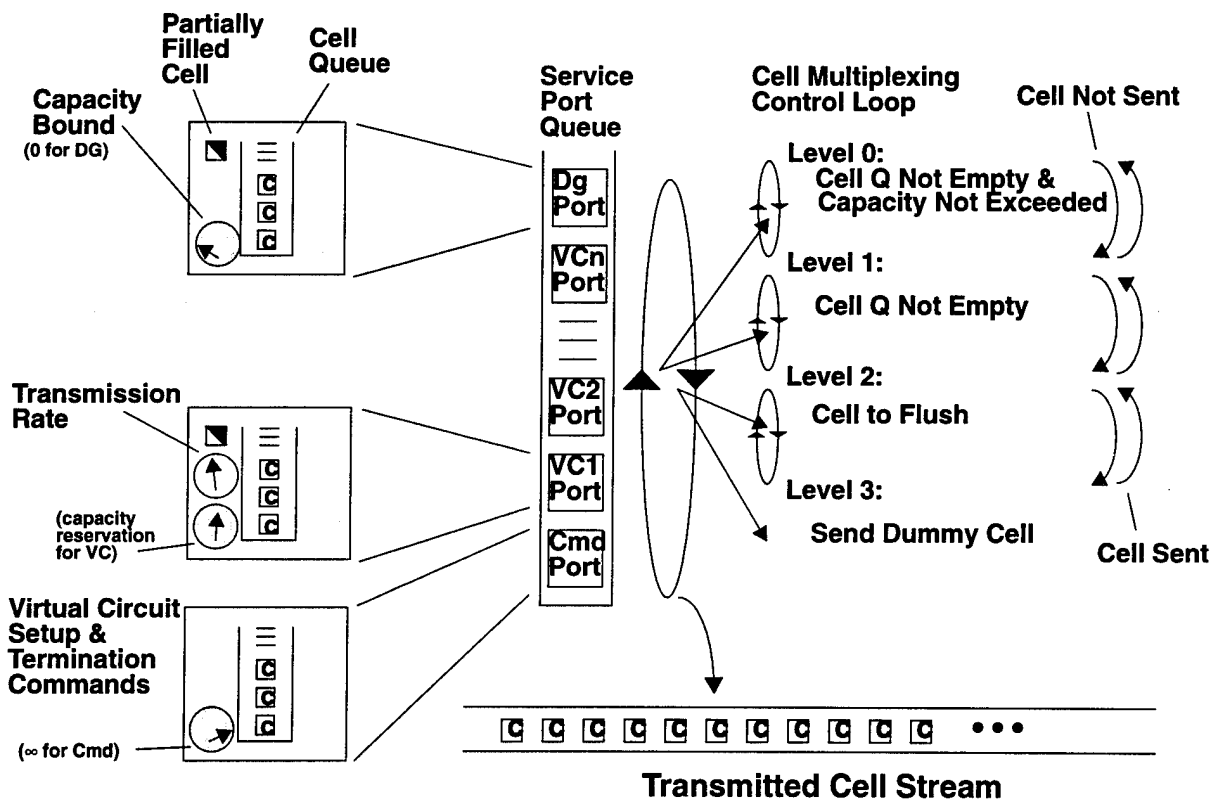


Figure 18. Multiplexing cells to create a transmitted cell stream.

all the capacity that the transmission system can provide, if needed. In actuality, DVI commands use only a very small fraction of the transmission capacity. VC Service Port's have capacity bounds that are set in accordance with user requests for capacity that have been honored by the DVI.

In addition, the DG and VC Service Ports may each have a partially filled cell that is in the process of being packed and has not yet been moved to the cell queue. It is moved to the cell queue when packing is completed or when the service port's *flush()* function is called. Also, the VC Service Port measures the user's transmission rate.

The service port entities just described are used by the cell multiplexing control loop to select the next cell for transmission and move it to the transmitted cell stream. The loop continually traverses the linked list of service ports in the Service Port Queue in an attempt to find a cell that meets the criterion for transmission. The search begins at Level 0, which uses the most restrictive test to determine if a cell can be sent. At Level 0, the cell queue must have a cell available to send and the capacity bound must not be exceeded if the cell is transmitted. The entire linked list of service ports is traversed using the Level 0 test. One cell is transmitted in turn from each service port that meets the Level 0 test criterion. If no cell is found that can be sent at Level 0, the level is incremented and the Service Port Queue is again traversed. At Level 1, the test criterion is relaxed so that a service port may send a cell if its cell queue contains a cell regardless of the port's capacity bound. If the entire Service Port Queue is traversed at Level 1 without finding a cell to send, the level is again incremented. At Level 2, partially filled cells are flushed and at Level 3 a dummy cell is sent. If traversal of the list at a given level results in the transmission of at least one cell, the level is decreased by 1.

The end result of the multiplexing protocol just described is that cells are removed and transmitted from eligible service ports in a round-robin fashion. The CMD Service Port and VC Service Ports that are not exceeding their capacity reservations are serviced first. The DG Service Port can only be serviced at Level 1 or higher, i.e., after the CMD and VC ports have received their allocated capacity, *if needed*. If VC ports don't use all the capacity they have been granted, then that unused capacity is automatically transferred to support the DG Service Port. Also, automatic flushing is performed as capacity permits.

5.7 DVI Software Design

The DVI software design takes advantage of the VxWorks multi-tasking operating system by using VxWorks tasks, sockets, and mailboxes. A brief description of some pertinent features of these VxWorks building blocks follows:

- **Task** - A VxWorks task is an independent program unit that has its own stack and program counter. The multi-tasking kernel creates an environment that gives the appearance of tasks executing in parallel when, in fact, at the kernel level only one task at a time is executed under control of the kernel's scheduler. The default protocol is preemptive priority-based scheduling, though round-robin scheduling may be selected as an alternative. One thing that a task does *not* have as part of its context is its own private address space. This is one of the ways VxWorks differs from Unix.
- **Socket** - VxWorks sockets provide a device driver type interface to either TCP or UDP and behave like their Unix counterparts. TCP provides a stream oriented connection trans-

port service. UDP provides a connectionless datagram transport service. The DVI uses stream sockets which, conceptually, are used no differently than the driver interface to an RS-232 port once a connection is established. The practical difference between the two is that sockets may create new connections under software control; RS-232 ports require manual recabling. A socket's blocking behavior is particularly critical to DVI design. A stream socket will block on a read to await data if none is available at the time the read command is issued. On a write, the socket will not block although it may return with an error or may write only part of the data if the task or process at the receiving end is not reading the data fast enough.

- Message Queue (i.e., Mailbox) - A mailbox is simply a FIFO queue provided by VxWorks to facilitate intratask communication. *msgQSend()* and *msgQReceive()* both take a time-out parameter that permits control of the blocking characteristics for both writes and reads.

Figure 19 presents a DVI software design using the VxWorks building blocks just described. DVImain creates a connection server socket, binds the socket to a well known port, and then listens for connections. When a connection is accepted, the connection server socket creates a stream socket to support the connection while DVImain spawns a DVIservice task to manage the server interface, allocate service to the client, and create a cell stream that carries the user's traffic. Messages sent by the client drive the execution of a DVIservice task, i.e., the task blocks on a read of the stream socket that it is servicing. Outputs of a DVIservice are the cell stream sent to the Multiplexer or messages sent to the client via the stream socket in support of the client/server interface.

DVImain also spawns the Multiplexer and Demultiplexer Tasks. The Multiplexer combines several homogeneous cell stream inputs into a single heterogeneous cell stream output, as previously described in section 5.6.3. In contrast to a DVIservice task, which is driven by its client, the Multiplexer task is driven by the RLC via the RLCmbx. The buffer capacity of the RLCmbx is set to a small value, e.g., two messages in the present implementation. Each message is 33 bytes in length, a 32 byte cell plus one byte of header for RLC control. By blocking the Multiplexer task processing loop on the write to the RLCmbx, the RLC can pace the execution of the Multiplexer task's decision making code. The alternative, i.e., allocating a larger buffer capacity to the RLCmbx, would allow a client (or clients) to temporarily exceed a sustainable transmission rate until the RLCmbx is filled. The problem in this approach is that the software making multiplexing decisions will not perform correctly if it is driven by a client generating a traffic burst. In that case, the only cell stream with cells available for multiplexing onto the output cell stream *at the time multiplexing decisions are made*, is the cell stream belonging to client generating the burst. Not permitting the RLCmbx to buffer the output cell stream avoids this potential problem.

Demultiplexing on the receive side of the DVI is handled by the Demultiplexer Task. Decoupling the transmit and receive sides of the DVI is not required for supporting half duplex communication, but it is essential for handling full duplex operation. Otherwise, a cell would have to be transmitted in order for a cell to be received and vice versa (or the *select()* function could be used). Having demultiplexed the stream of cells received via the RLC, the Demultiplexer must unpack the VC, DG, or CMD messages embedded in the demultiplexed cell stream and

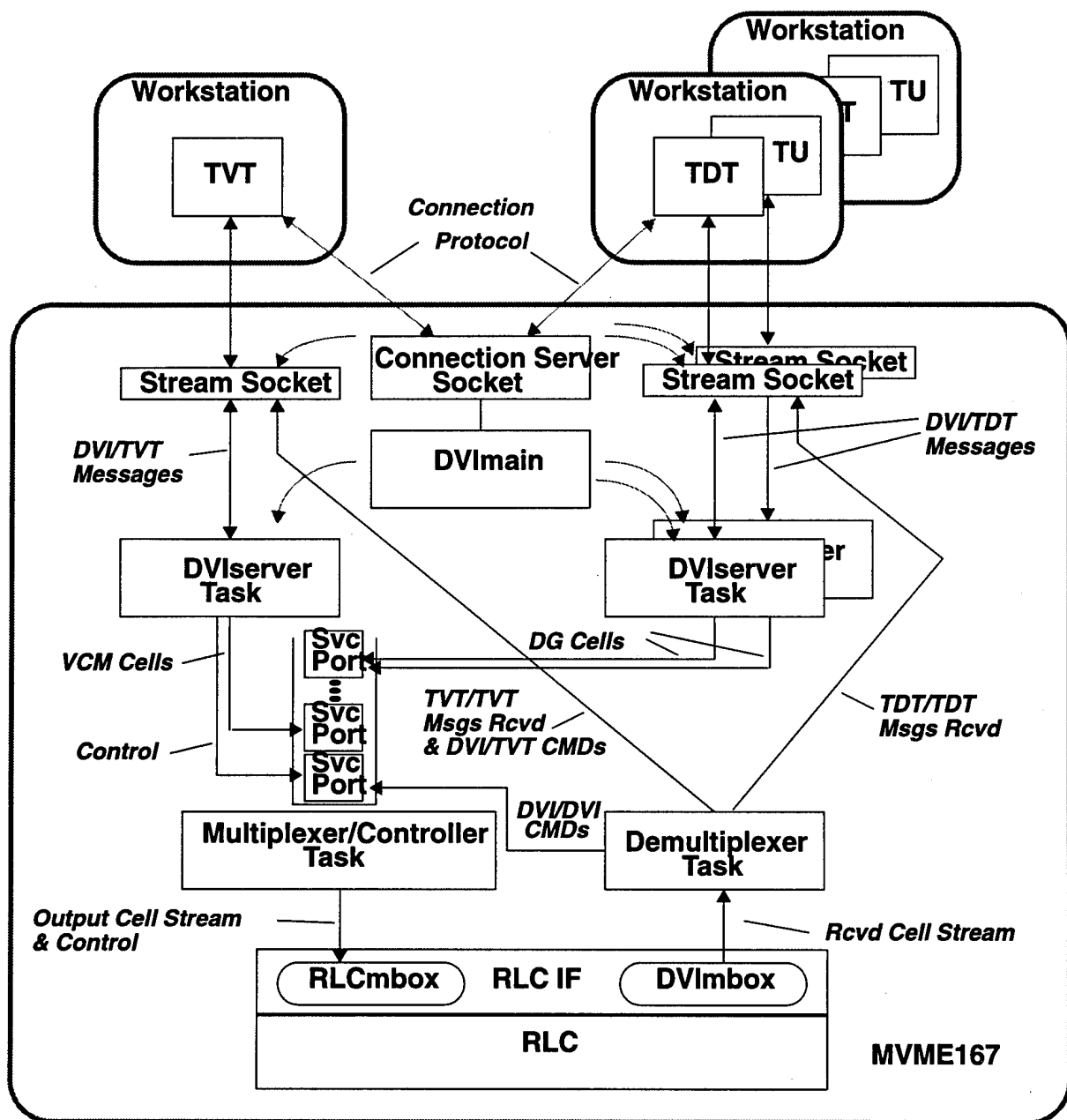


Figure 19. DVI software design.

deliver them to the proper recipient. When a CMD message is received, e.g., a *vcSetup* command, the Demultiplexer Task generates a *vcSetupRply* message that is passed to the Multiplexer Task (via the CMD Service Port) for transmission to the remote DVI that initiated virtual circuit setup. VC or DG messages that are received must be delivered to the recipient designated by the data-

gram's destination address or, in the case of VC traffic, the destination address that was given by the *vcSetup* message when the virtual circuit was established.

5.8 DVI Virtual Circuit Management

The DVI virtual circuit management protocols are the final element of the DVI design to be discussed in this document. The design presented here is the design implemented for the Phase I ATD system. For Phases II and III, the virtual circuit management protocols have been modified to support the Subnetwork Provider Interface (SNPI) that the CS uses to acquire services from the DVI and the MCA network (see figures 3 and 11). The key difference between the protocols is that the Phase I design creates virtual circuit connections between DVIs whereas the SNPI based protocols create connections between users. This difference is not as drastic as it may sound. Stated another way, the Phase I design sets up a circuit between DVIs based on the caller's request and then notifies the callee that a virtual circuit has been established. The callee then may decide to answer the call by using the virtual circuit thus established. In order to support the SNPI for Phases II and III, virtual circuit setup has been modified to pass the VC setup request to the user prior to establishing the virtual circuit. Thus, the callee must answer the call before the DVIs actually sets up the virtual circuit (see [3] and section 6.3).

Two basic types of virtual circuit service are supported in the Phase I DVI implementation - simplex and full duplex (i.e., *VCsmplx* and *VCdplx*). Figure 20 shows the simplex virtual circuit management protocol. When a caller issues a *vcSvcReq* to the local DVI requesting simplex virtual circuit service, the local DVI determines if it has a VCN available and can allocate enough capacity to honor the request. If it can, it returns a *svcGrntd* message to the caller and transmits a *vcSetup* message to the remote DVI. The remote DVI then notifies the callee that a simplex virtual circuit has been set up. The *svcNtfy* message also passes the caller's address and the type, priority, and capacity of the virtual circuit to the callee. Once service is granted, the callee may use *vcSnd* messages to supply data to the DVI for transmission. The DVI acknowledges every *vcSnd* command that it receives with a *vcStatus* message that tells the caller the status of the most recent *vcSnd*. The callee may terminate the virtual circuit by sending a *svcTerm* command to the DVI or by simply closing the TCP connection. In either case, a *vcTerminate* message is propagated to the remote DVI which, in turn, sends a *svcTermNtfy* message to the callee.

Figure 21 illustrates full duplex virtual circuit management. In many ways it resembles the simplex protocol except that both the caller and the callee may send data simultaneously and setting up the full duplex circuit requires an extra step. When the remote DVI receives the *vcSetup* message, it must have a VCN available and enough capacity to support a virtual circuit back to the caller. If it does, it sends a *vcSetupRply* back to the DVI where the call originated and notifies the callee that full duplex virtual circuit service has been established. Either user, i.e., caller or callee, may terminate the full duplex circuit.

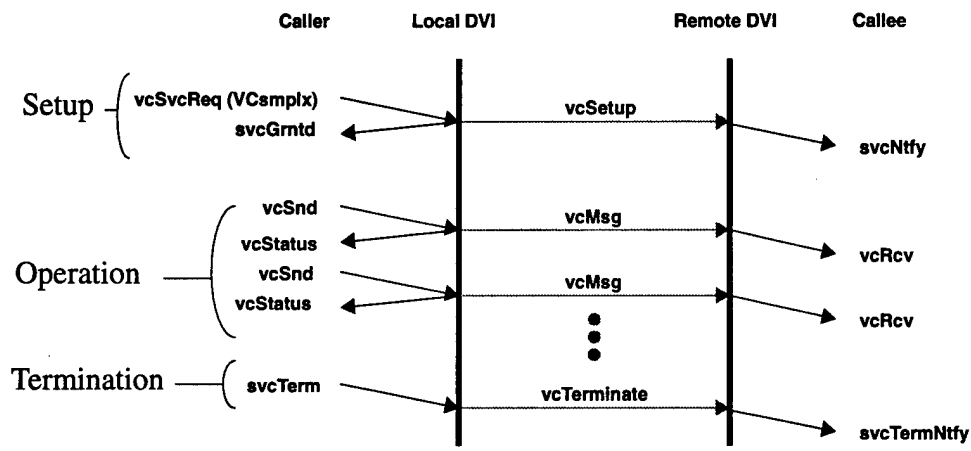


Figure 20. Setup, operation, and termination of a simplex virtual circuit.

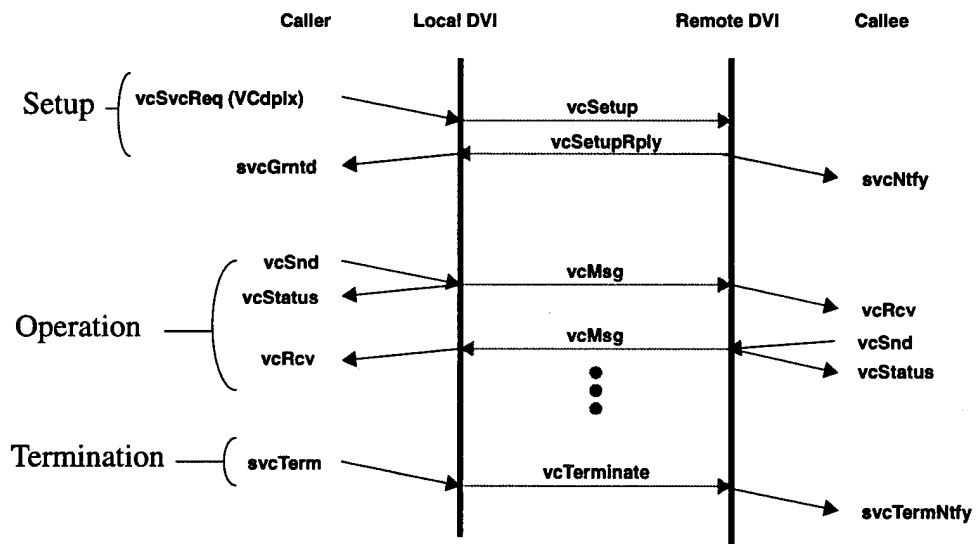


Figure 21. Setup, operation, and termination of a full duplex virtual circuit.

6.0 Appendix

6.1 Virtual Circuit Stream Service

A Virtual Circuit Stream (VCS) service has been designed as a companion to the Virtual Circuit Message (VCM) service. However, a decision has been made not to include VCS service in the Phase I implementation because VCS service will not be offered by the Phase II system. Providing a service in Phase I that is not offered in Phase II would negatively impact any user of that service when transitioning to the Phase II environment. The requirements for a VCS service are given in Table XIII and the message format portion of the VCS design is presented in figure 22. VC stream service, just like the DG and VC message services, does not guarantee byte reliability, but it does provide byte synchronization, i.e., it guarantees the number of bytes input to the virtual circuit stream (VCS) will be the same as the number of bytes output by the VCS. VCS

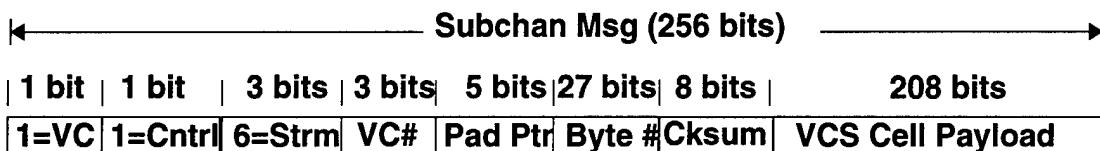


Figure 22. Cell format for a virtual circuit stream.

Table XIII: Virtual Circuit Stream Service Requirements

Service Type	Service Requirements
Virtual Circuit Stream	<ul style="list-style-type: none"> guaranteed throughput capacity guaranteed low latency (\leq max value) circuit is flushed automatically byte synchronization maintained (both number and order) - A null byte is inserted into the output byte stream for each byte lost by the VCS. bytes may contain bit errors

cells use six byte headers, which are lengthy compared to the one byte DG and the two byte VCM headers. On the other hand, VCS payloads don't carry additional message header overhead as do the VCM and DG cells.

6.2 Subnetwork Provider Interface (SNPI) Overview

The SNPI is documented in [3]. The intent here is to give a brief overview of the SNPI for the convenience of the reader who may not have a copy of [3] and to update the original design information given in section 5.5.

Table XIV: SNPI Messages

Message Type	SNPI Nomenclature	Sender->Receiver	Function
rgsReq	SN_RGS_REQ	client->DVI	client registers with server
rgsOK	SN_RGS_OK	DVI->client	registration successful
rgsFail	SN_RGS_FAIL	DVI->client	registration failed
dgSvcReq	SN_DG_SVC_REQ	client->DVI	client requests DG service
vcSvcReq	SN_VC_SVC_REQ	client->DVI	client requests VC service
svcNtfy	SN_SVC_IND	DVI->client	DVI notifies client that VC service has been set up to handle incoming call
svcRes	SN_SVC_RES	client->DVI	client responds to SN_SVC_IND (VC setup call)
svcTerm	SN_SVC_TERM	client->DVI	client terminates service
svcTermNtfy	SN_SVC_TERM_IND	DVI->client	DVI informs client that service has been terminated by remote client
svcGrntd	SN_SVC_CON	DVI->client	DVI grants service request
svcNtGrntd	SN_SVC_DEN	DVI->client	DVI denies service request
dgSnd	SN_DG_SND	client->DVI	client sends datagram to DVI for transmission
dgRcv	SN_DG_RCV	DVI->client	DVI delivers received datagram to client
dgPoll	SN_DG_POLL	DVI->client	DVI polls client for next datagram to send (polled service only)

Table XIV: SNPI Messages

Message Type	SNPI Nomenclature	Sender->Receiver	Function
dgStatus	SN_DG_STATUS	DVI->client	DVI acknowledges result of executing most recent dgSnd command
vcSnd	SN_VC_SND	client->DVI	client sends VC message to DVI for transmission
vcRcv	SN_VC_RCV	DVI->client	DVI delivers received VC message to client
vcStatus	SN_VC_STATUS	DVI->client	DVI acknowledges result of executing most recent vcSnd command
hdvRxOn	SN_RX_ON	client->DVI	client commands DVI to switch from transmit to receive mode
hdvTxHold	SN_TX_HOLD	client->DVI	client commands DVI to remain in transmit mode
hdvXoff	SN_XOFF	DVI->client	DVI response to a hdvTxHold that cannot be honored because DVI is in receive mode
hdvXon	SN_XON	DVI->client	if the DVI has issued a hdvX-off, it will issue a hdvXon when it returns to transmit mode
dviflush	SN_FLUSH	client->DVI	client commands DVI to pad and sent cell currently being packed
statusReq	SN_STATUS_REQ	client->DVI	client requests status information from DVI
statusRply	SN_STATUS_RPLY	DVI->client	DVI sends DVI status information to client
getPhyAddr	SN_GET_PHY_ADDR	client->DVI	get the physical, or node address, for this SC

Table XIV: SNPI Messages

Message Type	SNPI Nomenclature	Sender->Receiver	Function
setPhyAddr	SN_SET_PHY_ADDR	client->DVI	set the physical, or node address, for this SC
physAddrRply	SN_PHY_ADDR_RPLY	DVI->client	response to get or set physical address
snOK	SN_OK	DVI->client	response from SC to a number of control messages
snError	SN_ERROR	DVI->client	sent from DVI if invalid request was sent by client
vcRstReq	SN_VC_RST_REQ	client->DVI	reset a VC
vcRstInd	SN_VC_RST_IND	DVI->client	reset has occurred on VC specified
vcRstRes	SN_VC_RST_RES	client->DVI	client acknowledged reset of VC
vcRstCon	SN_VC_RST_CON	DVI->client	client is notified of VC reset completion
tstReq	SN_TST_REQ	client->DVI	send a test message
tstRply	SN_TST_RPLY	DVI->client	test message received
xidReq	SN_XID_REQ	client->DVI	send an XID message
xidInd	SN_XID_IND	DVI->client	XID message received
xidRes	SN_XID_RES	client->DVI	client response to XID message
xidRply	SN_XID_RPLY	DVI->client	XID reply delivered to client

6.3 Phase II (SNPI-based) Virtual Circuit Management

SNPI-based virtual circuit management is documented in [3]. The intent here is to give a brief overview of Phase II virtual circuit management for the convenience of the reader who may not have a copy of [3] and to update the original design information given in section 5.8. The virtual circuit setup protocols appear to be identical for simplex (VCsmplx), full duplex (VCdplx), and half duplex (VCswdplx) services and, at the protocol levels viewed in figures 23, 24, and 25, they are. However, at a lower level, the DVI differentiates between simplex and duplex (half or full) services. For simplex service, a single virtual circuit subchannel (defined by its vcn) is set up

at the local DVI. For duplex circuits, another virtual circuit subchannel is set up at the remote DVI to provide a return path to the local DVI.

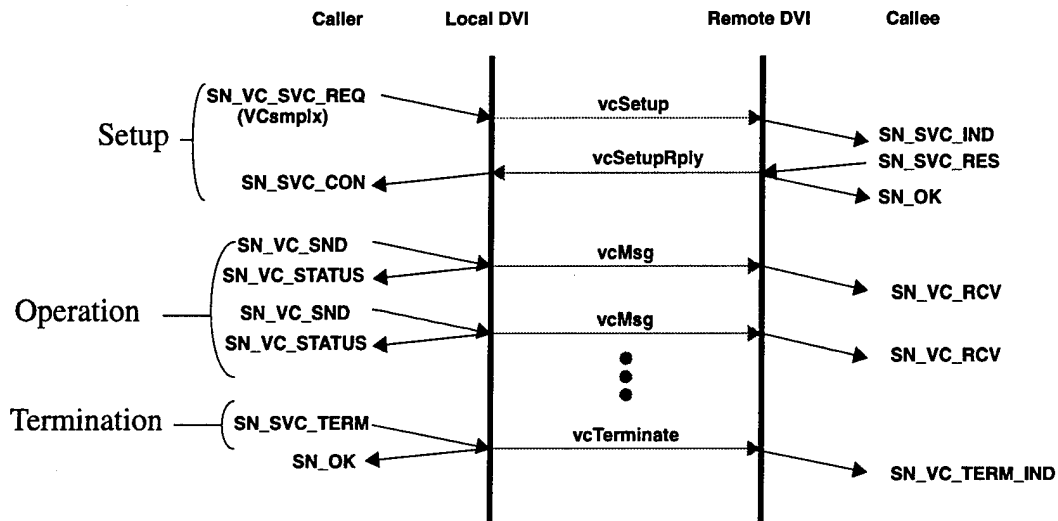


Figure 23. Setup, operation, and termination of a SNPI-based simplex virtual circuit.

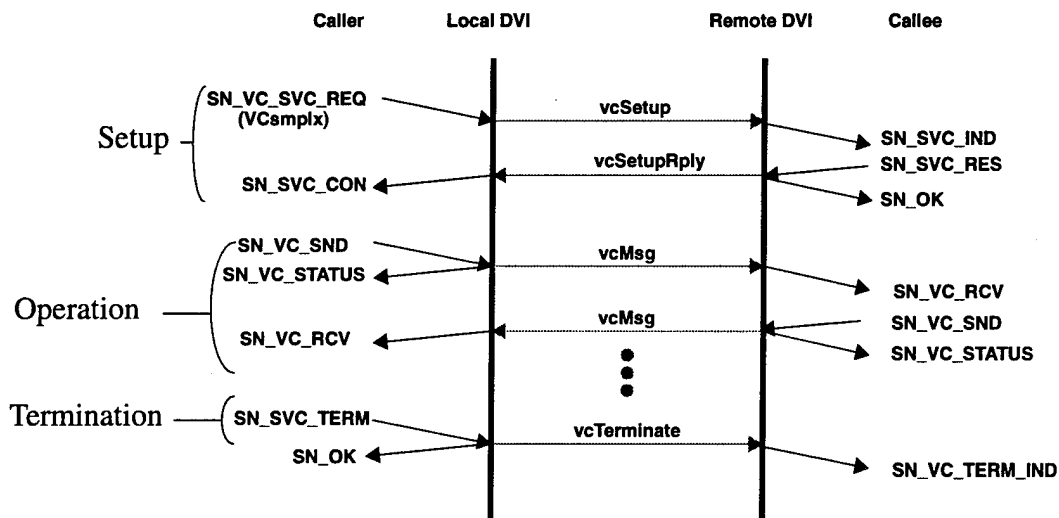


Figure 24. Setup, operation, and termination of a SNPI-based full duplex virtual circuit.

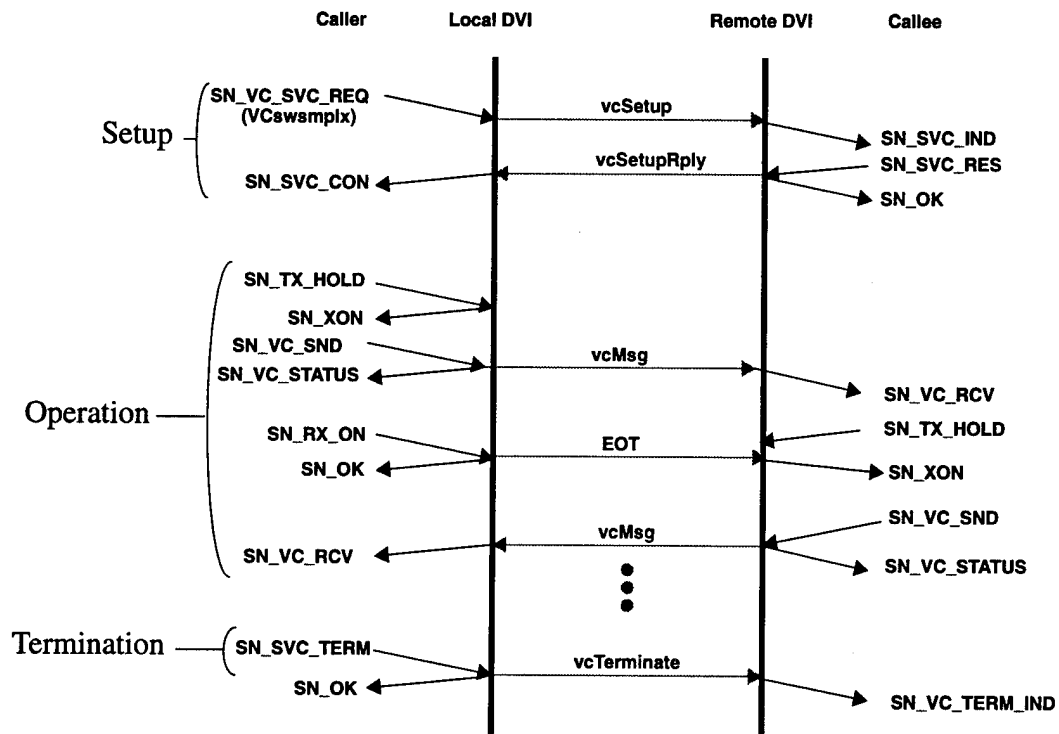


Figure 25. Setup, operation, and termination of a SNPI-based half duplex virtual circuit.